O‘ZBEKISTON RESPUBLIKASI
OLIY VA O‘RTA MAXSUS TA’LIM VAZIRLIGI

QARSHI DAVLAT UNIVERSITETI

FIZIKA-MATEMATIKA FAKULTETI

Amaliy matematika va informatika kafedrasi

“TIZIMLI DASTURLASH” fanidan

LABORATORIYA ISHLARI

5480100 – “Amaliy matematika va informatika”
bakalavr ta’lim yo‘nalishi III kurs talabalari uchun

1-laboratoriya ishi:

Translyatorlar. Dasturni kompilyatsiyalash usullari (6 soat)

2-laboratoriya ishi:

Assemblerda DOS uchun 16- va 32-bitli rejimda dasturlar tuzish (6 soat)

3-laboratoriya ishi:

Assemblerda Windows uchun 32-bitli dastur tuzish va uni kompilyatsiya qilish (4 soat)

4-laboratoriya ishi:

Assembler tilining makrovositalari bilan ishlash (4 soat)

5-laboratoriya ishi:

Dasturni boshqarish. Bir dasturni boshqa dastur orqali ishga tushirish (4 soat)

6-laboratoriya ishi:

COM- va EXE-fayllar bilan ishlash. EXE-faylni COM tipiga o‘tkazish (4 soat)

7-laboratoriya ishi:

Tizim uzilishlari. DOS va BIOS uzilishlari bilan ishlash (4 soat)

Laboratoriya ishlarini bajarishga doir umumiy ko‘rsatmalar

I. Laboratoriya mashg‘ulotini tashkil etish

Laboratoriya ishlari “Tizimli dasturiy ta’minot” kursini o‘rganish jarayonida belgilangan tematik reja asosida bajarilib boriladi va quyidagi qoidalarga amal qilinadi:

1. Tizimli dasturlarni tuzishda asosiy vosita sifatida assembler (istalgan versiya) yoki Si (istalgan versiya) dasturlash tillarining biridan foydalanish mumkin.
2. Tuziladigan dastur xususiyati va kompilyatsiyalash talabidan kelib chiqqan holda TASM, TASM5, MASM32, WASM, Turbo C.2.0, Borland C++ Builder 6 kabi dastur muharrirlarini ishlatish mumkin.
3. Ayrim hollarda DOS yoki Windows operatsion tizimidagi jarayonlarni tahlil qilish bilan bog‘liq masalalarga duch kelganda, ixtiyoriy vizual dasturlash tizimlarining biridan foydalanish mumkin.

II. Laboratoriya ishini bajarish tartibi

Barcha laboratoriya ishlari quyida ko‘rsatilgan talablar asosida bir xil tartibda amalga oshiriladi:

1. Masalaning qo‘yilishi va boshlang‘ich ma’lumotlar bilan tanishish;
2. Nazariy ma’lumotlar to‘plash va hisobot uchun tayyorlash;
3. Dastur tuzilishini loyihalashtirish;
4. Dastur tuzish;
5. Dastur matnini mashinaga kiritish;
6. Dasturni kompilyatsiya qilish;
7. Dasturda xatoga yo‘l qo‘yilgan bo‘lsa, tuzatish va tahlilini keltirish;
8. Natijani olish.
9. Yozma hisobot tayyorlash.

III. Hisobot mazmuni

Laboratoriya ishlarini topshirish uchun talaba hisobot tayyorlashi va hisobotda tubandagi bandlar to‘liq aks etgan bo‘lishi lozim:

1. Laboratoriya ishining mavzusi;
2. Ishning maqsadi;
3. Masalaning qo‘yilishi;
4. Mavzu va qo‘yilgan masalaga daxldor qisqacha nazariy ma’lumotlar;
5. Dastur matni;
6. Kompilyatsiyalash algoritmi matni;
7. Dastur natijasi (aynan ekrandagi tasviri bilan);
8. Xulosa.

1 – laboratoriya ishi

Mavzu: Translyatorlar. Dasturni kompilyatsiyalash usullari

Ishning maqsadi. Assembler tili translyatorlari va ularning xossalari bilan tanishish asosida kompilyatsiyalash qoidalari bo‘yicha amaliy bilim va ko‘nikmalar hosil qilish.

Masalaning qo‘yilishi. Assemblerda tuzilgan tayyor dastur matnini Turbo Assembler (TASM) va MakroAssemler (MASM) muharrirlari yordamida birma-bir translyatsiya qilish va ularning buyruq satri xossalari orasidagi farq bo‘yicha tahlil qilish.

Qisqacha nazariy ma’lumot. Assemblerda tuziladigan dasturlar mikroprotsessor imkoniyati va resurslarini to‘liq hisobga olgan holda amalga oshiriladi, masalan, 16-bit, 32-bit va hk. Dasturlarning tabiatiga bog‘liq ravishda ularni assemblerlash va bog‘lanish muharrirlari yordamida exe-faylga yig‘ish uchun turli kompilyatorlar ishlab chiqilgan. Masalan:

MASM (Macro Assembler) - Microsoft firmasining paketi. DOS bilan bir vaqtda Windows 9x/NT uchun assembler dasturlarini kompilyatsilashda ishlatiladi. Yangi versiyasi paketi MASM32 deb nomlanadi;

TASM (Turbo Assembler) - Borland firmasining mahsuloti. DOS dasturlarini kompilyatsiya qilishda qo‘llaniladi. Yangi versiyasi TASM5 yoki TASM5+ (TASM5Plus) deb ataladi;

WASM (Watcom Assembler) - Watcom firmasi tomonidan ishlab chiqilgan. DOS va Windows uchun tuzilgan dasturlarni kompilyatsiyalashda juda sodda interfeysga ega;

Lazy Assembler - TASM ning rivojlantirilgan versiyasi, yangi buyruq protsessorini o‘zida mujassamlashtirgan;

FASM (Flat Assembler) - MSDOS, Windows, Linux kabi operatsion tizimlarda ishlay oladigan, 16-, 32- hatto 64-razryadli protsessorlar uchun ham qulay ish muhitiga ega bo‘lgan kompilyator.

NASM (Netwide Assembler) - bu ham LINUX/BSD asosida ishlaydi;

YASM (Yet another assembler) – bu kompilyator ham NASM ning takomillashtirilgan versiyasi sanaladi.

HLA (High Level Assembly Language) - havaskor dastruchilar uchun yuqori darajali assembler tili kompilyatorlaridan biri sanaladi;

* * *

Yuqoridagi paketlar ichida eng ko‘p tarqalgani va foydalanishda qulay interfeysni taqdim etuvchi kompilyatorlardan asosiylari TASM, WASM hamda MASM lar sanaladi. Hamma kompilyatorlarning ham deyarli o‘ziga xos kamchiliklari mavjud. Shunday bo‘lsa-da, biz ushbu laboratoriya ishida TASM va MASM32 paketlari bilan ish yiritishni o‘rganamiz. Chunki, ular dasturga ilova qilinadigan kutubxona fayllarining boyligi, ishlashda ravon va tezlik samaradorligining yuqoriligi, turli modullardan foydalanish imkonoyatining mavjudligi bilan boshqalaridan ajralib turadi.

TASM (Turbo Assembler) - Borland firmasi tomonidan yaratilgan bo‘lib, u 16/32-bitli mikroprotsessorlar uchun yozilgan dasturlarni kompilyatsiyalashda qo‘llaniladi. Bu kompilyator bevosita MS DOS muhitida ishlaydi. Turbo Assemblerning bir vaqtda ikkita versiyasidan foydalanish mumkin - **tasm.exe** va **tasmx.exe**. Juda katta modullarni assemblerlashtirishda TASM ni ishlatish kerak. Chunki, TASM versiyasi TASMX ga qaraganda tezroq ishlaydi.

Avvalo, Assemblerda tuziladigan dasturlarni qayta ishlash (exe-faylga aylantirish) uchun uni translyatsiya jarayoniga tayyorlash kerak. Dasturni bajariluvchi faylga translyatsiya qilish va uni protsessorning joriy holatida qadamba-qadam bajarilishini nazorat qilish hamda o‘rganish jarayoni to‘rt bosqichni o‘z ichiga oladi.

1-bosqich. Dastlabki (boshlang‘ich) dastur matni tayyorlanadi va u biror **xxxx.asm** fayl ko‘rinishida tegishli katalogda (albatta TASM paketi katalogida) saqlab qo‘yiladi;

2-bosqich. TASM translyatori orqali dastur assemblerlanadi, natijada **xxxx.obj** kengaytmali obyektli fayl hosil qilinadi. (**Obyektli fayl** - bu dasturning ikkili-kod ko‘rinishida tasvirlanishidir).

Agar dastur bir nechta mustaqil fayllardan (alohida modullardan) iborat bo‘lsa, assemblerlash jarayoni bu dasturlarning har biri uchun alohida bajariladi.

Agar dastlabki dastur matnida avvaldan xatoga yo‘l qo‘yilgan bo‘lsa yoki translyatsiya qilish jarayonida biror xatolikka yo‘l qo‘yilsa, u holda assemblerlashdan keyin ekranga bu xatoliklar haqidagi xabar chiqadi. Xatolar bartaraf qilingan taqdirda translyatsiyani takroran amalga oshirish zarur bo‘ladi.

3-bosqich. Dasturni yig‘ish (kompanovkalash). Bu ish **Turbo Linker** yig‘uvchisi (bog‘lanish muharriri) orqali amalga oshiriladi va

nihoyat **xxxx.exe** yoki **xxxx.com** kengaytmali bajariluvchi fayl hosil qilinadi.

4-bosqich. Tuzilgan dasturning xotira maydonida joylashgan holati, uning haqiqiy mashina kodida ifodalanishi va buyruq formatini o‘rganish uchun uni TD muharririda ishga tushirish.

Misol tariqasida istalgan matn muharririni ishga tushiramiz. Masalan, Windowsning “Блокнот”, “AkelPad” kabi ichki matn muharriri, yoki DOS muhariri, yoki agar mavjud bo‘lsa Assembler dasturlarini qayta ishlash va sozlashga mo‘ljallangan maxsus “Text_AsmVDP” integrallashgan paketidan foydalanishingiz mumkin. Demak, kompyuterga quyidagi dastur matnini kiritamiz va uni **Hello.asm** deb nomlab C:\ diskning TASM katalogida saqlab qo‘yamiz (agar haqiqatan ham TASM paketi shu diskda joylashgan bo‘lsa).

Dastur matni quyidagicha:

```
;savol.asm
.MODEL small
.stack 256

.data

Promt DB '-3 soni -5 sonidan kattami? (Ha/Yo`q - Y/N): $'
No    DB 13,10,'Javobingiz noto`g`ri!',13,10,'$'
Yes   DB 13,10,'Javobingiz to`g`ri!',13,10,'$'

.code

Start: mov ax,@data     ;Ma'lumotlar segmenti adresini DS ga o`rnatish
       mov ds,ax
       mov dx,OFFSET Promt       ;So`rov-xabari
       mov ah,9          ;Satrni ekranga chiqarish -DOS funksiyasi
       int 21h
       mov ah,1       ;DOS funksiyasi - klaviaturadan simvolni kiritish
       int 21h
       cmp al,'y'     ;y?
       jz  IsYes      ;Ha,katta
       cmp al,'n'     ;n?
       jz  IsNo       ;Yo`q, kichik

IsNo:  mov dx,OFFSET No
         jmp SHORT Disp
```

```
IsYes: mov dx,OFFSET Yes

Disp: mov ah,9
      int 21h

Exit:  mov ax,4C00h    ;DOS funksiyasi - dasturdan chiqish
       int 21h
End start          ;Dastur tugadi
```

Endi **savol.asm** nomli ushbu dasturni qadamba-qadam translyatsiya qilish jarayonini qarab chiqamiz. Buning uchun MS DOS muhitida **C:\TASM** katalogiga kiramiz (bu **cd tasm** buyrug‘i bilan bajariladi). MSDOS buyruqlar satrida quyidagi

```
C:\TASM> tasm savol.asm
```

yozuvini kiritamiz va “Enter”ni bosamiz. Agar xatolik bo‘lmasa, ekranda quyidagi xabar chiqadi:

```
TurboAssemblerVersion3.2i Copyright(c) 1988,1992Borland International
Serial No:   Tester:

Assembling file:   savol.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  455 k
```

Bunda:

(1) - Turbo Assembler versiyasi, yili, firmasi nomi va mualliflik huquqi haqidagi xabar
(2) - Serial nomeri
(3) - assemblerlashtirilgan fayl nomi: **savol.asm**
(4) - xatolar haqida xabar**: yo‘q**
(5) - ogohlantirish xabari: **yo‘q**
(6) - o‘tishlar: **1**
(7) - qolgan xotira kattaligi: **455 k**.

Demak, natijada **savol.obj** fayli (dasturning oraliq formasi) hosil bo‘lganligini ko‘rish mumkin. Endi dasturni Turbo Linker bog‘lanish

muharriri yordamida kompanovka qilish (yig‘ish) qoldi. Buning uchun yana MS DOS buyruqlar satrida

```
C:\TASM> tlink hello.obj
```

buyrug‘ini terish yetarli. Shunday qilib, **savol.exe** bajariluvchi fayl ham hosil bo‘ldi, buni quyidagi xabardan bilish mumkin:

```
Turbo Link Version 5.1 Copyright (c) 1992 Borland International
C:\TASM>
```

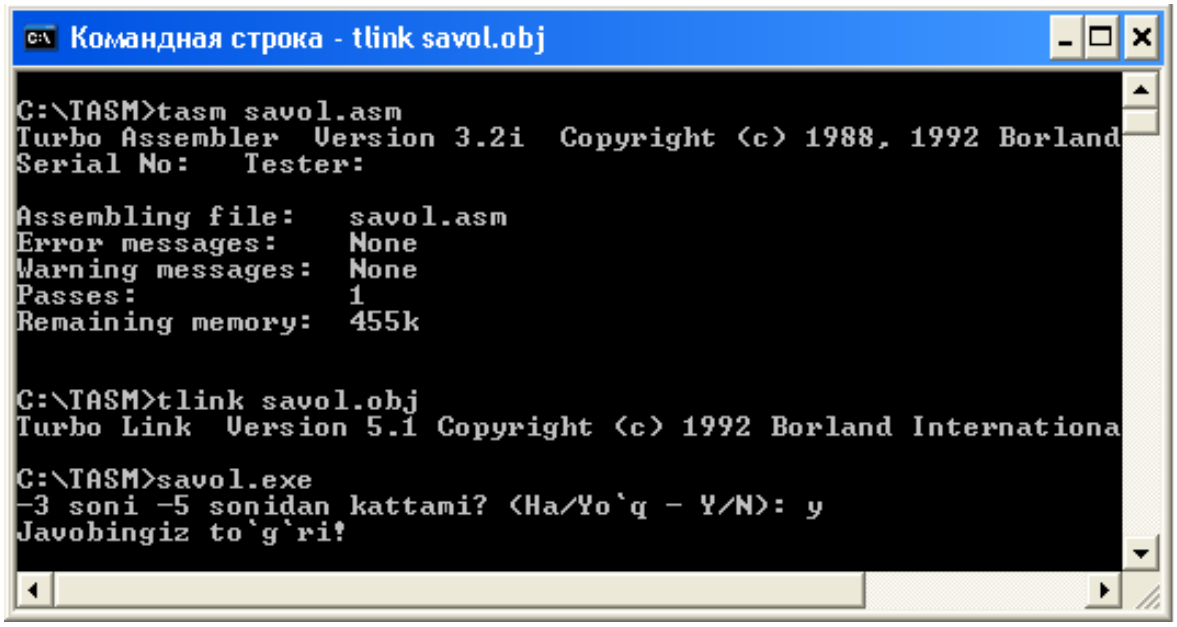
Endi **savol.exe** dasturni ishga tushirib, ekranda quyidagi dastur natijasini ko‘rish mumkin:

```
-3 soni -5 sonidan kattami? (Ha/Yo`q - Y/N):
```

Dastur klaviaturadan Y yoki N klavishni bosihni kutadi. Agar siz Y harfini kiritsangiz, darhol keyingi xabar ekranda ikkinchi satrdan paydo bo‘ladi, ya’ni

```
Javobing to`g`ri!
```

Tushunish oson bo‘lishi uchun dastur natijasining ekrandagi to‘liq tasvirini beramiz.



Dasturchi uchun boshlang‘ich matnda xatoga (leksik yoki sintaktik) yo‘l qo‘yish tabiiy holat. Bunday paytlarda fayllarni yuqoridagi singari qadamba-qadam kompilyatsiya qilish ancha vaqtni olishi mumkin. Shu maqsadda bu ishni tezlashtirish maqsadida ***.BAT**

fayllardan foydalanish anchagina qulaylik tug‘duradi. Umuman olganda **.bat** kengaytmali faylni hosil qilish qiyinish emas. Uni ixtiyoriy matn muharriri orqali hosil qilish mumkin. Yaxshisi, qadamba-qadam bajarib ko‘ramiz:

1. Ixtiyoriy matn muharrirlaridan birini oching (masalan: Bloknot, WordPad, AkelPad, MS DOS yoki FAR ichki muhariri);
2. Matn muharriri oynasida quyidagilarni xuddi ko‘rsatilgandek 2 ta satrni tering;

```
tasm savol.asm
tlink savol.obj
```

4. Endi faylga **savol.bat** nomini berib, TASM katalogining ichida saqlang;
5. Agar dastlabki **savol.asm** faylga tuzatish kiritgan yoki uni biroz o‘zgartirgan bo‘lsangiz, u holda ekranda quyidagi buyruqni tering:

```
C:\TASM>savol.bat
```

Dasturni ishlatib ko‘rishingiz mumkin.

Bu ishni 2-usul bilan ham amalga oshirish mumkin. Buning uchun yuqoridagi kabi matn muharriri oynasida quyidagi satrlarni kiriting:

```
tasm %1.asm
tlink %1.obj
```

Faylga esa, masalan **dmake.bat** deb nom bering va TASM katalogida saqlang. Faqat shu narsaga e’tibor berish kerakki, bunda siz qo‘ygan fayl nomi TASM katalogidagi birorta ham **.exe** kengaytmali fayl nomi bilan bir xil bo‘lmaslik kerak (masalan, **tasm.bat** deb qo‘yib bo‘lmaydi).

Shunday qilib, **dmake.bat** faylimizni ishlatib ko‘ramiz, ya’ni MS DOS da faqat **dmake savol** matnini yozsangiz kifoya:

```
C:\TASM\Dmake savol
```

Bu usul birinchisiga qaraganda ancha qulayroq, chunki birinchisi aynan bitta fayl uchun, ikkinchisi esa barcha fayllar uchun umumiy vazifani bajaradi. Tajribali dasturchilar o‘zlari istagandek, bu dastur-faylni kerakli buyruqlar bilan boyitishlari mumkin.

Umuman olganda, biz hozircha **tasm.exe** va **tlink.exe** buyruqlar satrining eng sodda ko‘rinishidan foydalandik. Lekin aslida ularning sintaksisi umumiy ko‘rinishda quyidagicha bo‘ladi:

```
Syntax: TASM [options] source [,object] [,listing] [,xref]
```

Assemblerlash yoki kompanovkalash jarayonida **Tasm.exe** yoki **Tlink.exe** bilan birga yoziladigan turli xossalar (opsiyalar) dan foydalanish mumkin. Xossalar bir yoki bir nechta harflar hamda ularning yonida og‘ma chiziq orqali yoziladi. TASM ning barcha xossalari ro‘yxatini chiqarish uchun MS DOS buyruqlar satrida

Tasm.exe /h

ni terib “Enter” ni bosamiz (bu amal **tlink.exe** uchun ham xuddi shunday bajariladi):

Xossalari:

/a, /s - Alphabetic or Source-code segment ordering (segmentlarni alifbo bo‘yicha yoki dastlabki kod tarkibi bo‘yicha tartibga solish)

/e, /r - Emulated or Real floating-point instructions (Qo‘zg‘aluvchan nuqtali haqiqiy sonlar ustida bajariladigan amal kodlarini generatsiya qilish)

/h, /? - Display this help screen (ekranda ushbu ma’lumot xabari chiqadi)

/iPATH - Search PATH for include files (qo‘shiladigan fayllar PATH bilan berilgan yo‘ldan axtariladi)

/l, /la - Generate listing: l=normal listing, la=expanded listing (listingni chiqarish: */*- odatdagi listing; */a* - kengaytirilgan listing)

/ml, /mx, /mu - Case sensitivity on symbols: **ml**=all, **mx**=globals, **mu**=none (identifikator harflarini registrda ajratish: */ml*-hammasini, */mx*-global, */mu*-ajratilmaydi)

/mv# - Set maximum valid length for symbols (idenfikatorlar nomlarining maksimal uzunligini qo‘yish)

/mn - Allow n multiple passes to resolve forward references (translyator jarayonidagi o‘tirshlar sonini belgilaydi. Zarurat bo‘lganda 5 tahacha o‘tish sonini o‘rnatish mumkin. Agar n qo‘yilmasa, translyator 1 ta o‘tishni bajaradi)

/n - Suppress symbol tables in listing (listing faylda identifikatorlar jadvalini chiqarish haqida ko‘rsatma)

/os, /o, /op, /oi - Object code: standard, standard w/overlays, Phar Lap, or IBM (Overley kodining turli variantdagi generatsiyasi)

/p - Check for code segment overrides in protected mode (Himoyaviy rejimda ishlash)

/q - Suppress OBJ records not needed for linking (mkompanovka bosqichida kerak bo‘lmaydigan ortiqcha ma’lumotlarni obyekt kodidan chiqarib tashlash)

/t - Suppress messages if successful assembly (shartli assemblerlashda xatolik haqidagi xabarlardan boshqa xabarlarni chiqarmaslik)

/w0, /w1, /w2 - Set warning level: w0=none, w1=w2=warnings on (turli ogohlantiruvchi xabarlarni generatsiya qilish yoki qilmaslik haqida ko‘rsatma)

/x - Include false conditionals in listing (shartli assemblerlashning barcha obyektlarini listingga qo‘yish)

/z - Display source line with error message (yuzaga kelgan xatoliklarni satrma-satr o‘qiydi)

/zi, /zd, /zn - Debug info: zi=full, zd=line numbers only, zn=none (zi - TD.EXE tuzatish muharriri uchun kerakli ma’lumotlarni obyekt faylga qo‘shish; zd - satr nomerlari haqidagi ma’lumotni qo‘shish; zn - TD.EXE tuzatish muharriri uchun kerakli ma’lumotlarni obyekt faylga qo‘shishni ta’qiqlash)

Endi **Tlink.exe** opsiyalarini (xossalarini) keltiramiz. Umumiy ko‘rinishda yozilish sintaksisi quyidagicha:

Syntax: TLINK objfiles, exefile, mapfile, libfiles, deffile

Xossalari:

/m - Map file with publics (Файл Карты с публикой - ma’lumotli birlashtirishlar xaritasi faylini yaratish)

/l - Initialize all segments (Инициализируйте все сегменты - barcha segmentlarni faollashtirish)

/L - Specify library search paths (Определите библиотечные пути поиска - Kutubxona fayllariga yo‘lni topish)

/n - No default libraries (Нет встроенных библиотек - tashqi kutubxona fayli yo`q)

/c - Case significant in symbols (h значимый в символах – belgilarning ishora qiymatini o‘rnatish)

/o - Overlay switch (Оверлейный ключ - Overley kalitini o`rnatish)

/P[=NNNNN] - Pack code segments (Кодовые сегменты Упаковки)

/ye - Expanded memory swapping (Расширенный свопинг памяти - kengaytirilgan xotira svopingi)

/e - Ignore Extended Dictionary (Проигнорируйте Расширенный Словарь - kengaytirilgan lug‘atni e’tiborga olmaslik)

/t - Create COM file (same as /Tdc) (Создайте файл COM - COM fayl yaratish)

/C - Case sensitive exports and imports (Чувствительный экспорт и импорт Случая)

/Txx - Specify output file type (Определите выходной файловый тип - natijaviy fayl tipini aniqlash)

/Tdx - DOS image (default) (ОБРАЗ DOS (невыполнение) - DOS tipidagi fayl)

/Twx - Windows image (ОБРАЗ WINDOS - Windows tipli fayl)

(third letter can be c=COM, e=EXE, d=DLL)

(третье буква может быть c=COM, e=EXE, d=DLL)

(uchinchi harf c bo`lsa COM, e bo`lsa EXE, d bo`lsa DLL)

/x - No map file at all (Никакой файл карты совсем - .map fayl yaratmaslik haqida ko‘rsatma)

/l - Include source line numbers (Включите исходные номера линии - dastlabki kodning satr tartib raqamini kiritish)

/s - Detailed map of segments (Подробная карта сегментов - segmentlarning to`liqroq xaritasi)

/d - Warn if duplicate symbols in libraries (Предупредитесь если двойные символы в библиотеках – agar kutubxonada bir xil belgilar uchrasa ogohlantirish)

/3 - Enable 32-bit processing (Допустимая 32-битовая обработка - 32-bitli rejimni qo`llash)

/v - Full symbolic debug information (Полная символическая отладочная информация - tuzatish tahririda simvollarni to`liq aniqlash)

Shunday qilib, ushbu xossalar yordamida dastur uchun turli ko‘rinishdagi translyatsiyalarni olish mumkin.

Endi MASM32 da translyatsiya qilish jarayoni bilan tanishamiz. MASM32 paketi aslida 32-bitli dasturlarni translyatsiya qilishga mo‘ljallangan bo‘lsa-da, ammo u 16-bitlik dasturlarni ham translyatsiya qilish uchun kerakli vositalarni o‘zida qo‘llash imkoniyatini beradi. MASM32 da dasturni assemblerlash uchun **ml.exe**, linklash uchun esa **link.exe** (32-bitli) yoki **link16.exe** (16 bitli) dasturlaridan foydalaniladi. Tushunish oson bo‘lishi uchun yuqoridagi dasturimizni MASM32 da kompilyatsiyalashni amalga oshrib ko‘ramiz (**savol.asm)**:

1. Assemblerlash:

```
C:\Masm32\bin\ml.exe /c savol.asm
```

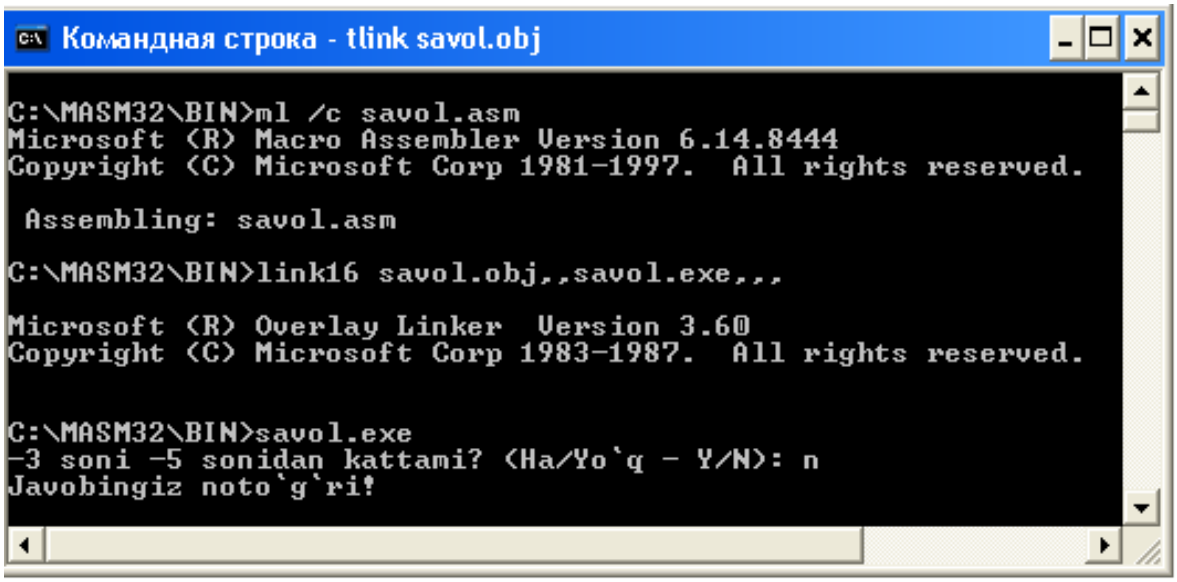
Bu yerda, **/c** - opsiyasi (xossasi) linkovkasiz (bog‘lanish muharririsiz) assemblerlash jarayonini bildiradi.

2. Linklash:

```
C:\Masm32\bin\link16 savol.obj,,savol.exe,,,
```

Bu yerda, biz yuqorida aytganimizdek 16-bitli **link**’dan foydalandik, chunki dastlabki kod matni 16-bitli rejimda tuzilgan (bu haqda keyingi mashg‘ulotda batafsil ma’lumot beriladi).

Shunday qilib, yana **savol.exe** fayli hosil bo‘ldi. Dastur natijasini o‘zingiz ko‘ring:



Endi **ml.exe** va **link.exe** bilan birga beriladigan eng asosiy va juda ham ko‘p ishlatiladigan ba’zi xossalarini keltiramiz. **ml.exe** ning umumiy sintaksisda berilishi:

ML [/options] filelist [/link linkoptions]

Xossalari:

/c - Assemble without linking **(**ассемблирование без линковки - linkovkasiz assemblerlash);

/COFF - generate COFF format object file (генерировать объектный файл в COFF формате - obyektli faylni COFF formatga generatsiya qilish);

/Fo<file> - Name object file (имя объектного файла - obyekt fayl nomi)

link.exe sintaksisining umumiy holda ko‘rinish:

LINK [options] [files] [@commandfile]

Xossalari:

/SUBSYSTEM:{...} - Qism sistema, bunda dastur bajarilishi uchun quyidagi muhitlardan biri tanlanishi kerak bo‘ladi:

NATIVE|WINDOWS|CONSOLE|WINDOWSCE|POSIX;

/LIBPATH:path - *.lib kutubxona fayliga yo‘l ko‘rsatish;

/DLL - bunda hosil qilinadigan oxirgi natijaviy fayl .exe tipda emas, balki .dll tipda bo‘ladi;

/I <name> - *.inc faylga yo‘l ko‘rsatiladi va hk.

Shunday qilib, MASM32 da dasturni translyatsiya qilish xuddi TASM’dagi singari oddiy, ammo ustunlik jihatlari anchagina. Bunda ham dasturni tez kompilyatsiyalash uchun ***.bat** faylni bir marta tuzib olib, ko‘p marta foydalanish mumkin.

Topshiriqlar

Quyida berilgan yozuvlarga mos translyatsiya xossalari ma’nosiga ko‘ra o‘rganib chiqilsin va aynan qanday maqsadda qo‘llanilishi konkret dasturiy misollar orqali yozma hisobotda keltirilsin:

Вариант	Vazifa
1	Tasm.exe /ml xxxxx.asm
2	Tlink.exe /t xxxxx.obj
3	ml.exe /c /coff xxxxx.asm
4	Link.exe /subsystem:console xxxxx.obj
5	Tasm.exe /n xxxxx.asm
6	Tasm.exe /zi xxxxx.asm
7	Tasm.exe // xxxxx.asm
8	Tasm.exe /*la* xxxxx.asm
9	Link16 xxxxx.obj,,xxxxx.exe,,,,
10	Exe2bin xxxxx.exe xxxxx.com
11	Tlink /x /t xxxxx.obj
12	Tlink /Tdx /s xxxxx.obj
13	Tasm /m /z xxxxx.asm
14	ml.exe /c /Cp xxxxx.asm
15	Link /subsystem:windows xxxxx.obj

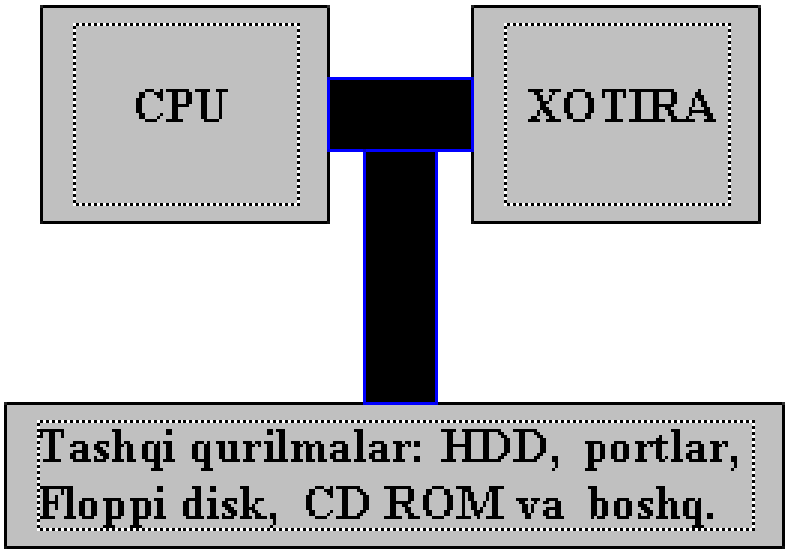
2 – laboratoriya ishi

Mavzu: Assemblerda DOS uchun 16- va 32-bitli rejimda dasturlar tuzish.

Ishning maqsadi. Assembler tili translyatorlari va ularning xossalari bilan tanishish asosida kompilyatsiyalash qoidalari bo‘yicha amaliy bilim va ko‘nikmalar hosil qilish.

Masalaning qo‘yilishi. Assemblerda tuzilgan tayyor dastur matnini Turbo Assembler (TASM) va MakroAssemler (MASM) muharrirlari yordamida birma-bir translyatsiya qilish va ularning buyruq satri xossalari orasidagi farq bo‘yicha tahlil qilish.

Qisqacha nazariy ma’lumot. DOS yoki Windows uchun Assembler tilida dasturlar tuzishni o‘rganish uchun avvalo, protsessor va xotira tuzilmasini chuqurroq tushunish lozim. Uning yetarlicha sodda sxemasini keltiraylik:



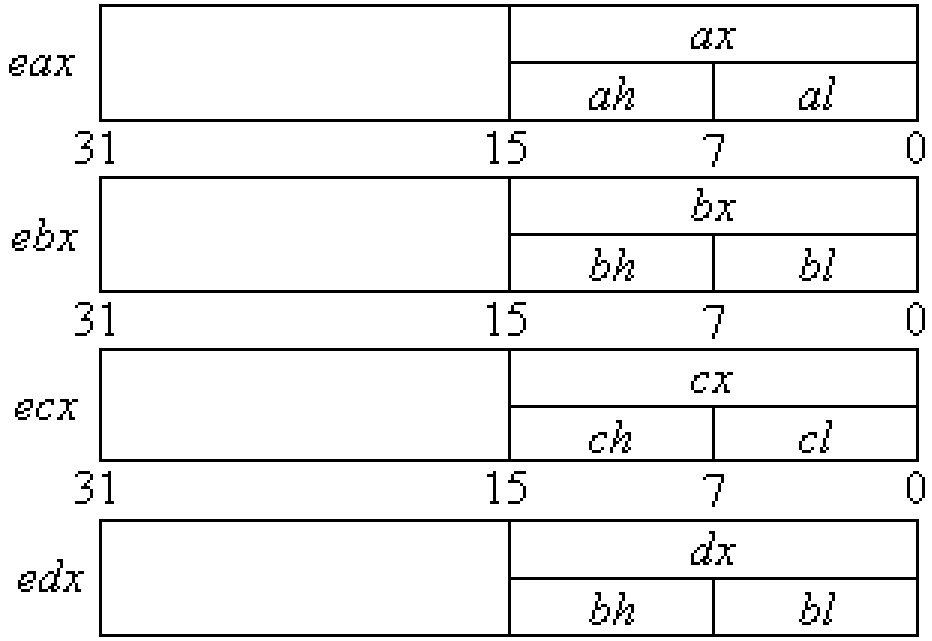
Ma’lumki, protsessor asosiy obyekt - bu registrlar hisoblanadi. Registrlar 8, 16, 32(64) bo‘lishi mumkin. Biz ko‘pincha umumiy qilib, x86-protsessorlar, deb gapiramiz. x86-protsessor deganda, bu protsessor 8086, 80186, 80286, 80386, 80486, 80586 (Pentium) va shu kabi protsessorlardan biri bo‘lishi mumkin.

Shuni farqlash lozimki, 8086-80286 protsessorlar 16-bitli, 80386 dan boshlab undan keyingilari 32-bitli protsessorlardir.

80386 protsessorning dasturiy modeli bilan tanishamiz.

32-bitli 80386 protsessorlarda umumiy vazifali registrlar *eax, ebx, ecx, edx* bilan, 16-bitli protsessorlarda esa bu registrlar *ax, bx, cx,*

dx bilan belgilanadi. Bunda har bir registr 2 qismga ajratiladi: kichik qism - *al, bl, cl, dl* va katta qism - *ah, bh, ch, dh*.



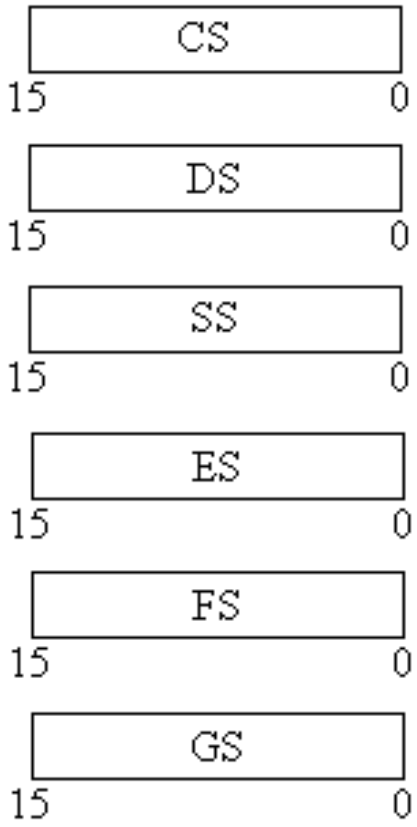
Umumiy vazifali registrlar asosan, ma’lumotlar ustida matematik amallar, taqqoslash amallari, xotiraga yozish va shu kabi maqsadlarda qo‘llaniladi.

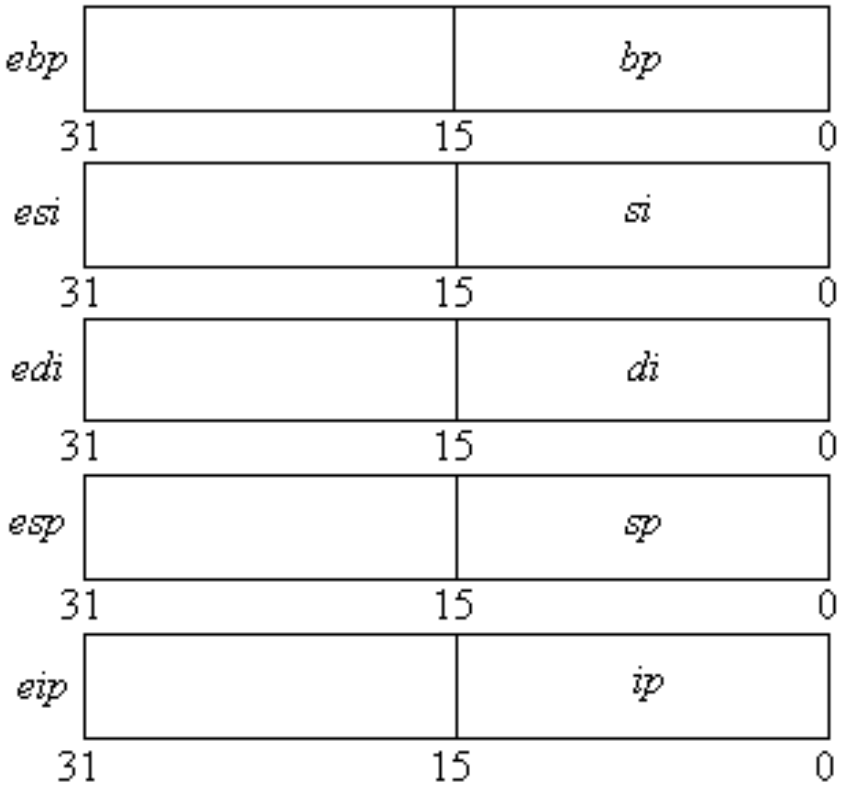
CS, DS, ES, FS, GS va SS - bular 16 bitli segmentli registrlar bo‘lib, ular protsessorda **“offset:segment”** adresining birinchi yarmini o‘z ichiga oladi.

Offset registrlari - *ebp, esi, edi, esp* va *eip* lar hisoblanadi. Bu registrlar 32-bitli bo‘lib, ularning quyidan birinchi yarmini *bp, si, di, sp* va *ip* registrlar (16-bitli) deb qarash mumkin.

Intel 80386 - protsessorlaridan boshlab undan keyingilari 2 xil rejimda ishlaydi: **oddiy** va **himoyaviy** rejimda. Himoyaviy rejimda ishlovchi protsessorlar 32-bitli tuzilmaga ega bo‘lgan registrlar asosida ishlaydi. Masalan, Win32 himoyaviy rejimda (p80386+), DOS esa oddiy rejimda ishlaydi. DOS da 32-bitli registrlar yo‘q.

DOS da va Windows 3.x da xotira segmentlardan tarkib topgan. Bu segment bir xil 64 Kb hajmdan iborat bo‘lib, bunda har bir registr adresi segment formatida beriladi: **offset**. Segment o‘zining mos tartib raqami bilan, offset esa bu segmentdagi siljish adresi orqali belgilanadi.





Xotira				
Segment 1 (64 Kb)	Segment 2 (64 Kb)	Segment 3 (64 Kb)	Segment 4 (64 Kb)	va hokazo

Bu yerda 16-razryadli dastur tuzish maqsadida qo‘llaniladiga xotira birligi haqida gap ketyapti. Jadvaldan ko‘rinib turibdiki, umumiy xotira hajmi 64 Kb dan teng taqsimlangan segmentlardan tarkib topgan. Endi bitta segmentning o‘zini olib qaraylik:

Segment (64 Kb)				
Siljish 1	Siljish 2	Siljish 3	Siljish 4	va hokazo

Ushbu 16 lik sonlar orqali 0030:4012 yozuvni misol sifatida olib tahlil qilaylik: bunda 30 segment adresini, 4012 esa siljish adresini bildiradi.

Siljish tushunchasini yana ham chuqurroq tushunish uchun quyidagi buyruqlar ketma-ketligiga e’tibor beramiz:

```
mov cx, 100
mov ax, 200
push cx
push ax
xor cx, ax
add cx, 400
mov dx, cx
pop bx
pop cx
```

Stekda bajarilgan ushbu amallar xotirada qanday yuz berishini tahlil qilishga o‘tamiz, bunda 1-jadval stek hozircha nollar bilan to‘ldirilgan, hali hech qanday amal bajarilmagan holatni anglatadi:

Siljish	1203	1204	1205	1206	1207	1208	1209	120A	120B
Qiymat	00	00	00	00	00	00	00	00	00
					ESP				

Mov ax, 4560h
Push ax

Siljish	1203	1204	1205	1206	1207	1208	1209	120A	120B
Qiymat	00	00	60	45	00	00	00	00	00
			ESP						

Mov cx, FFFFh
Push cx

Siljish	1203	1204	1205	1206	1207	1208	1209	120A	120B
Qiymat	FF	FF	60	40	00	00	00	00	00
	ESP								

pop dx

Siljish	1203	1204	1205	1206	1207	1208	1209	120A	120B
Qiymat	FF	FF	60	40	00	00	00	00	00
					ESP				

dx endi **4560FFFFh**.

Shunday qilib, assembler buyruqlari orqali operandlarni adreslashtirishda odatda, **real** va **himoyaviy** rejimdan foydalaniladi. Real rejimda siljish har doim 16-bitli bo‘ladi, 32-bitli siljish bilan adreslashtirish (himoyaviy rejimda), masalan, Windows uchun dastur tuzishda qo‘llanilishi mumkin.

Ma’lumki, DOS da bajariluvchi dasturlar asosan ikki xil tipda bo‘ladi: **exe** va **com** tipida. Boshqacha aytganda, masalan, **format.exe** yoki **format.com** tipida bo‘ladi. Quyida 16-bitli rejimda **.com** tipli dastur tuzish orqali fikrimizni davom ettiramiz, bunda operandlarni to‘g‘ri adreslashtirish usulidan foydalanamiz. Qo‘shiluvchi sonlarni **p1** va **p2** deb belgilab, dastur tarkibida bevosita ularning p1=39, p2=12 qiymatlarida hisoblashni amalga oshiramiz. Dastur natijasi ekranga 51 (39 va 12 sonlarning yig‘indisi) sonini chiqarishdan iborat.

Demak, quyida berilgan dastur matnini biror matn muharriri orqali kompyuterga kiritib, unga **add16.asm** deb nom beramiz va TASM katalogida saqlaymiz.

Dastur matni quyidagicha:

```
;add16.asm
; 10 lik sanoq sistemasida ikki son yig`indisini hisoblash

        .model tiny
        .code
        org 100h
start:
```

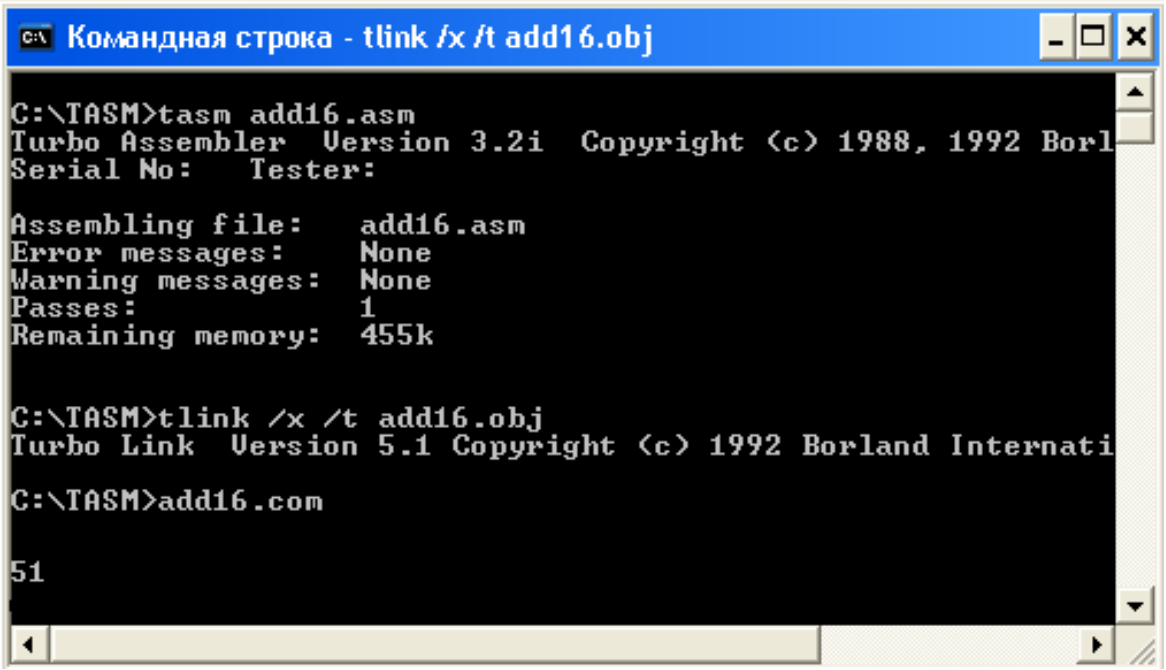
```
        mov AL,CS:P1+1
        add AL,CS:P2+1
        aaa
        mov CS:SUM+1,AL
        mov AL,CS:P1
        adc AL,CS:P2
        aaa
        mov CS:SUM, AL
        add BYTE PTR CS:SUM,48
        add BYTE PTR CS:SUM+1,48
        mov DX, OFFSET SUM
        P1   DB '39'
        P2   DB '12'
SUM  DB '  ', 13, 10, '$'
        mov ah,9
        int 21H
        ret
END start
```

Endi MS DOS buyruqlar satrida kompilyatsiyalashni amalga oshiramiz. Natija com-fayldan iborat bo‘ladi.

Kompilyatsiyalash (TASM da):

```
tasm add16.asm
tlink /t /x add16.obj
```

Dastur natijasi quyidagicha bo‘ladi:



Endi xuddi shu amalni bajaradigan dasturni 32-bitli rejimda tuzishga harakat qilamiz. Masala tushunarli bo‘lishi uchun bunda ham 39 va 12 sonlarining yig‘indisini hisoblash dasturini tuzamiz.

Dastur matni quyidagicha:

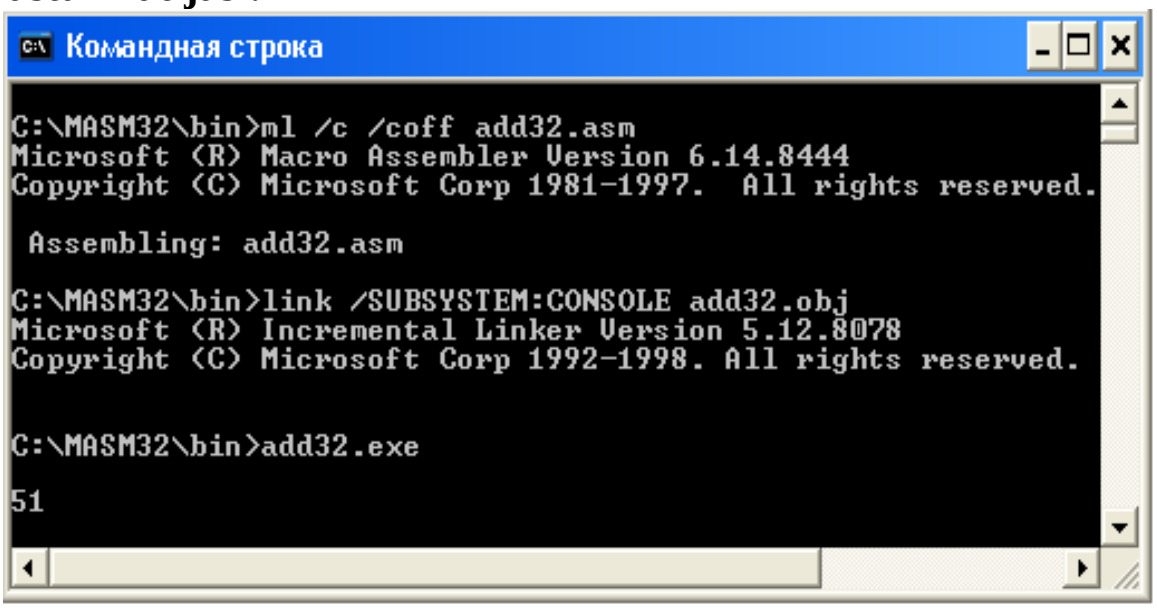
;add32.asm

```
.386
.model  flat, stdcall
option  casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc

includelib   \masm32\lib\user32.lib
includelib   \masm32\lib\kernel32.lib

BSIZE equ 15

.data
ifmt   BYTE "%d",00
stdout   DWORD ?
cWritten DWORD ?

.data?
buf    BYTE  BSIZE dup(?)

.code
start:
invoke  GetStdHandle, STD_OUTPUT_HANDLE
mov stdout, eax
mov eax,39
add eax,12
invoke  wsprintf, ADDR buf, ADDR ifmt,eax
invoke WriteConsoleA, stdout, ADDR buf, BSIZE, ADDR cWritten,NULL
invoke ExitProcess, 0
end start
```

Kompilyatsiyalash (MASM32 da):

```
ml /c /coff add32.asm
link /SUBSYSTEM:CONSOLE add32.obj
```

Dastur natijasi:



Xulosa. DOS asosida tuzgan birinchi **add16.asm** dasturiimiz matnidan ham ko‘rinib turibdiki, bunda dastur com-fayl tipida qurishga mo‘ljallangan. Chunki dastur avvalida xotiraning **TINY** modelidan foydalanilsh haqida ko‘rsatma berilgan. **TINY** xotira modeli asosan DOS uchun COM tipidagi dasturlarni tuzishga mo‘ljallangan. Ammo bu yerda EXE tipli dasturni ham tuzish mumkin edi, faqat buning uchun xotiraning **flat** modelidan hamda bir nechta segmentlardan foydalanish kerak bo‘lardi. Bu unchalik qiyin masala emas. Chunki, dasturda foydalanishda asosiy ishtirok etadigan kodlar alohida ravishda CS - kodlar segmentiga, ma’lumotlar esa DS - ma’lumotlar segmentiga yoziladi va hokazo. COM - dasturlarda esa barcha ma’lumotlar va kodlar yuqoridagi kabi bitta mantiqiy segmentda beriladi. Bu haqda keyingi mashg‘ulotlarda batafsil to‘xtalamiz.

Odatda assemblerda tuziladigan har bir dastur tarkibidagi har bir satrda yoziladigan buyruqlar uchun uning ma’nosini tushuntirib boruvchi izoh so‘zlar (kommentariyalar) yoziladi. Lekin bu kommentariyalar translyator tomonidan e’tiborga olinmaydi. Dastur matni bir qarashda murakkab tuzilishni ifodalamasligi uchun ataylab kommentariyalarni turshirib qoldirdik.

Topshiriqlar

Berilgan topshiriqlarga mos 16- va 32-bitli dasturlar tuzilsin va dastur natijalari tahlili hisobotda talab qilingan bandlar bo‘yicha yozma shaklda keltirilsin.

Вариант	Vazifa
1	MS DOS versiyasini aniqlash
2	Tizimli soatni o‘qish va ekranga chiqarish
3	Ekrandagi mavjud tasvir nusxasini printerga chiqarish
4	Joriy sanani aniqlash va ekranga chiqarish
5	Disket (A:) ga belgi (Label) qo‘yish
6	Disket (A:) 0-sektor ma’lumotlarini o‘qish
7	Tizimni qayta yuklash
8	ASCII belgilaridan ixtiyoriy bittasini ekranga chiqarish
9	Qattiq disk (C:) parametrlarini aniqlash
10	Dasturni rezident holda qoldirib tugatish
11	Ekranga faqat bitta belgini chiqarishni ta’minlash
12	Klaviatura Ctrl-Break tugmasi bosilishini qayta ishlash
13	Klaviatura Ins tugmasi bosilishini qayta ishlash
14	Operativ xotira hajmini aniqlash
15	Komputerga tarmoq qurilmalari o‘rnatilgan yoki o‘rnatilmaganligini aniqlash

3 – laboratoriya ishi

Mavzu: Assemblerda Windows uchun 32-bitli dastur tuzish va uni kompilyatsiya qilish.

Ishning maqsadi. Assembler tilida Windows ilovalari uchun 32-bitli rejimda dastur tuzilishini o‘rganish va dasturni Masm32 kompilyatori orqali bajariluvchi faylga kompilyatsiyalash qoidalari bo‘yicha amaliy bilim va ko‘nikmalar hosil qilish.

Masalaning qo‘yilishi. Sarlavha satrida va markazida ixtiyoriy matnni hosil qiluvchi Windows muloqot oynasini dasturlash va kompilyatsiyalash.

Qisqacha nazariy ma’lumot. Ma’lumki, Windowsda bajariladigan barcha amallar odatda oynalardan foydalangan holda bajariladi. Assemblerda Windows uchun tuzuladigan har bir dastur asosida shu an’anani davom ettirish mumkin. Dastur tarkibida ishtirok etadigan buyruqlar va ularning sintaksisi DOS ga nisbatan Windowsda sezilarli tafovutlarga ega. Bu farqlarni keyinchalik yo‘l-yo‘lakay

ko‘rib o‘tamiz. Endi dasturning asosiy tuzulmasiga e’tiborni qaratamiz. Oldin dasturning asosiy “skeleti”ni tuzib olamiz:

```
.386

.model flat, stdcall

option casemap:none

;(WinAPI funksiyalarini chaqirish, makros (makroaniqlov))

.const

...

.data

...

.data?

...

.code

...

start:

...

invoke ExitProcess,0

end start
```

Windows uchun yoziladigan barcha 32-bitli dasturlarda standart ravishda quydagi 3 ta satr mavjud bo‘lishi shart:

```
.386
.model flat, stdcall
option casemap: none
```

Bunda birinchi satr biz tuzadigan dasturimiz uchun 386-protsesorni tanlaganimizni bildiradi (8086-, 80186-, 80286-

protsessorlar 16-bitli, 80386-, 80486-, 80586-, va undan keyingi protsessorlar 32-bitli).

Ikkinchi satr esa Windows uchun 32-bitli dastur tuzishda xotiraning FLAT modelidan foydalanayotganimizni ifodalaydi.

Uchunchi satrdagi yozuv orqali katta va kichik harflar turlicha ekanligi, ya’ni bir-biridan farqlanishi kerakligi haqida kompilyatorga ko‘rsatma beriladi.

Windows ilovalarini dasturlashda ko‘pincha tashqi resurslardan foydalanishga to‘g‘ri keladi. Bunday resurslardan eng asosiysi Windowsning DLL-kutubxonasida joylashgan tizimli funksiyalardir. Windows uchun dastur tuzishda ulardan foydalanish - odatda, Windows tizimli funksiyalarni chiqarish deb ataladi. Tizimli standart funksiyalar chaqirilishni quyidagi satrlar misolida ko‘rish mumkin:

```
include \MASM32\INCLUDE\windows.inc
include \MASM32\INCLUDE\masm32.inc
include \MASM32\INCLUDE\gdi32.inc
include \MASM32\INCLUDE\user32.inc
include \MASM32\INCLUDE\kernel32.inc

includelib \MASM32\LIB\masm32.lib
includelib \MASM32\LIB\gdi32.lib
includelib \MASM32\LIB\user32.lib
includelib \MASM32\LIB\kernel32.lib
```

Aynan bitta dastur uchun bu kutubxonalarning barchasiga ehtiyoj yo‘qdir, ammo umumiy holda ularni dastur tarkibiga xuddi shablon singari kiritib qo‘yish ham zarar qilmaydi, chunki ular eng muhim tizimli funksiyalar kutubxonasi sanaladi. Tuziladigan dasturlarning tabiatidan kelib chiqib, taxminan 2500 dan ortiq bunday tizimli funksiyalardan foydalanishimiz mumkin.

Bu satrlardan keyin o‘zgaruvchilar satrini qo‘yish, ya’ni o‘zgaruvchilarni e’lon qilish bo‘limi boshlanadi. O‘zgaruvchilar uch xil tipda bo‘ladi: konstantalar, dastlabki qiymatiga ega bo‘lgan o‘zgaruvchilar, vaqtincha qiymatlarga ega bo‘lmasdan turub, dastur davomida yo‘l-yo‘lakay foydalaniladigan o‘zgaruvchilar.

Konstantalar, dasturda **`.const`** satridan keyin joylashadi. Masalan:

```
.const
alfa equ 5
```

Ammo, bizning dasturimizda bu bo‘limga hozircha ehtiyoj yo‘q. Shuning uchun dasturimizda bu satrlardan foydalanmasligimiz mumkin.

Keyingi bo‘lim dastur bajarilishi mobaynida o‘z individual qiymatlari bilan ishtirok etadigan o‘zgaruvchilarni e’lon qilish bo‘limi, ya’ni **.data** bo‘limidir (bu direktiva orqali berilgan ma’lumotlar aslida DS – ma’lumotlar segmentida joylashadi). Bu bo‘limda e’lon qilinadigan o‘zgaruvchilar 2 tipda bo‘ladi: **Byte** va **Dword**. Byte tipli o‘zgaruvchilar asosan catrli (matn) ma’limotlarni e’lon qilishda ishlatiladi, Dword tipli o‘zgaruvchilar esa 4-baytdan iborat bo‘lib, asosan sonlarni ifodalashda foydalaniladi. Bizning misolimizda byte tipli ikkita: ourmessage va ourtitle o‘zgaruvchilar qo‘llaniladi.

```
.data
ourmessage db "Bu Windows uchun dastur!",0
ourtitle db "Assemblerda dasturlash!",0
```

Bi yerda ourmessage db buyrug‘i orqali biz dasturlayotgan muloqot oynasining sarlavha satridagi yozuv, ourtitle db buyrug‘i orqali esa oyna markazida beriladigan yozuv ifodalanadi.

Keyingi bo‘lim - **.data?** o‘zgaruvchilarni e’lon qilish bo‘limi bo‘lib, ular quydagicha e’lon qilinadi:

```
.data?
undefinedvariable dd?
undefinedbyte db?
```

Bu bo‘limdan bizning dasturda foydalanilmaydi. Shuning uchun uni dastur tarkibida akslantirish shart emas.

Va, nihoyat, kodni yozish bo‘limiga o‘tamiz. Kodlar .code satridan keyin **start:** belgisi bilan boshlab yoziladi va **end start** belgisi bilan tugatiladi.

Boshqa asosiy bajariladigan kodlar qatorida Windows tizimli funksiyalarini chaqirish ham aynan shu kodlar bo‘limi tarkibida bajariladi. Windowsning bu tizimli funksiyalarini API funksiyalari deb ham atashadi. Biz dasturimizda aynan Windows oynasidan foydalanishini ko‘zda tutgan holda uning **MessageBox** standart

funksiyasini chaqirishimiz kerak bo‘ladi. Bu funksiya aslida suzuvchi oynani yaratish va uni ekranga chiqarish vazifasini bajaradi. Qo‘shimcha ma’lumot sifatida uning mazmunini ko‘rib qo‘yish foydadan xoli emas:

```
int MessageBox(

HWND hWnd, // handle of owner window

LPCTSTR lpText, // address of text in message box

LPCTSTR lpCaption, // address of title of message box

UINT uType // style of message box

);
```

Bunday funksiyalarni chaqirish uchun assemblerda **INVOKE** buyrug‘idan foydalaniladi. Bunda parametrlar vergullar bilan ajratib yoziladi. Shunday qilib, kodlar bo‘limida:

```
INVOKE MessageBox,0,ADDR ourmessage,ADDR ourtitle,MB_OK
INVOKE ExitProcess, 0
```

kodlarni yozamiz.

Bu yerda jarayonni tugatish va boshqaruvni operatsion sistemaga qaytarish uchun ExitProcess funksiyadan foydalanilmoqda. Demak:

```
.code
start:
invoke MessageBox,0,ADDR ourmessage,ADDR ourtitle,MB_OK
invoke ExitProcess,0
end start
```

Shunday qilib, dasturimizda ishlatiladiga barcha parametrlarni ham aniqlab oldik. Endi uni quyidagicha ifodalaymiz:

```
.386
.model flat, stdcall
option casemap :none

include \MASM32\INCLUDE\windows.inc
include \MASM32\INCLUDE\masm32.inc
```

```
include \MASM32\INCLUDE\gdi32.inc
include \MASM32\INCLUDE\user32.inc
include \MASM32\INCLUDE\kernel32.inc

includelib \MASM32\LIB\masm32.lib
includelib \MASM32\LIB\gdi32.lib
includelib \MASM32\LIB\user32.lib
includelib \MASM32\LIB\kernel32.lib

.data
ourmessage db "Bu Windows uchun dastur!",0
ourtitle db "Assemblerda dasturlash!",0

.code
start:
invoke MessageBox,0,ADDR ourmessage,ADDR ourtitle,MB_OK
invoke ExitProcess,0
end start
```

Tuziladigan dasturimiz bor-yo‘g‘i ana shundan iborat. Endi uni fayl sifatida nom berib (masalan **p00.asm**) Masm32\bin katalogida saqlaymiz. Dasturni kompilyatsiyalash DOS da ham, Windowsning o‘zida, QEditor oynasida ham amalga oshirilishi mumkin.

Windowsda kompilyatsiyalash:

1) **QEditor.exe** ishga tushiriladi;
2) Hosil bo‘lgan oynaning **Open** menyusi orqali **p00.asm** yuklanadi;
3) Menyudan **Project** ga kirib, **Build All** buyrug‘i bosiladi.

Shunday qilib, bizda endi quyidagi uchta fayl mavjud:

p00.asm – dastlabki fayl,
p00.obj – obyektli fayl,
p00.exe – bajariluvchi fayl (dastur).

Endi **p00.exe** ni Windowsdan yoki DOS dan ishga tushirib, dastur natijasini ko‘rishingiz mumkin.

Dastur natijasi:

MS DOS muhitida kompilyatsiyalash:

1-qadam: 

2-qadam: MS DOS buyruqlar satrida ketma-ket **cd masm32 (Enter)** va **cd bin (Enter)** buyruqlarni terish orqali Masm32\bin katalogiga kiriladi;

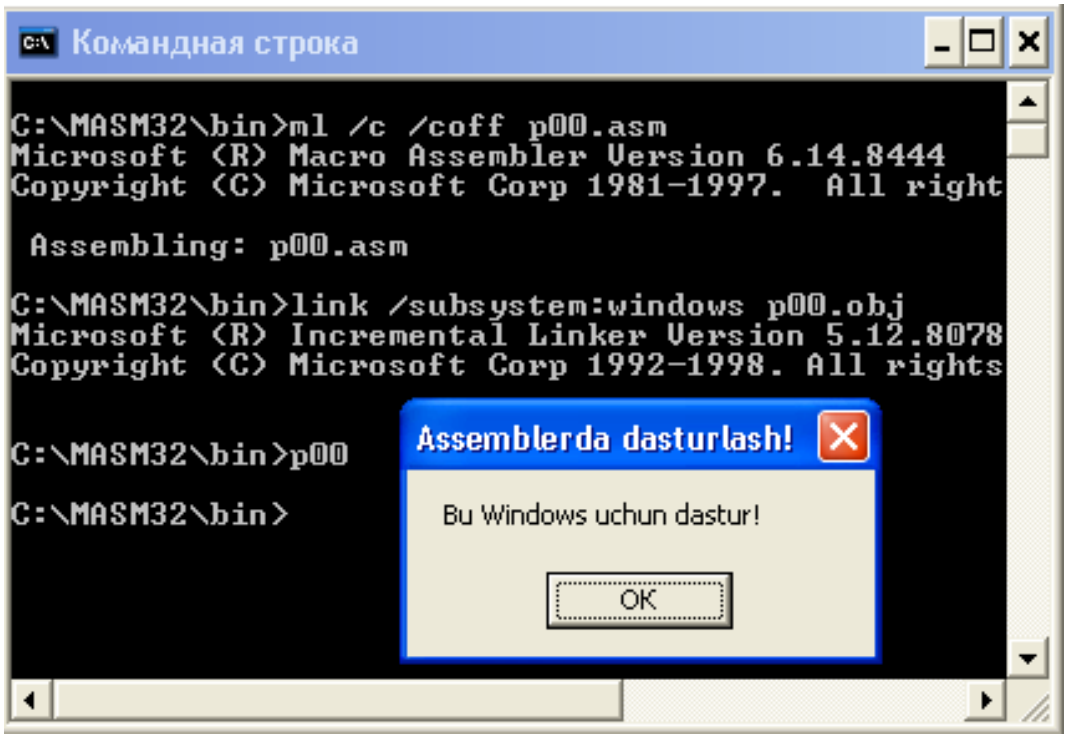
3-qadam: Quyidagi buyruq orqali dasturni assemblerlaymiz:

ml /c /coff p00.asm

4-qadam: 32-bitli bog‘lanish muharriri yordamida **p00.obj** faylni **p00.exe** – bajariluvchi faylga kompilyatsiyalaymiz:

link /subsystem:windows p00.obj

Dastur natijasi:



Xulosa. Tuzilgan dastur 32-bitli rejimga asoslangan bo‘lganligi uchun uni MASM32 da kompilyatsiya qilish jarayonini qarab chiqdik. Lekin turli kompilyatorda bu ishni amalga oshirish mumkin. Faqatgina buyruq satridagi parametrlar (kompilyator xossalari) bir-biridan birozga farq qilishi mumkin. Tushunishda qulaylik tug‘dirish maqsadida eng sodda ko‘rinishdagi ushbu dasturdan foydalanildi.

Topshiriqlar

1. MS DOS buyruqlar satri mazmunini ekranda hosil qiluvchi dastur tuzilsin.
2. Qalqib chiqadigan izoh so‘zli oynani hosil qilish dasturi tuzilsin.
3. Ekranda matn hosil qiluvchi konsol ilovali dastur tuzilsin.
4. Windowsda (a∗b+c)/d ifoda qiymatini oynada hosil qiluvchi dastur tuzilsin.
5. Joriy mikroprotsessor chastotasini oynada hosil qiluvchi dastur tuzilsin.
6. Joriy disk parametrlarini ko‘rsatuvchi grafik ilovali dastur tuzilsin.
7. Operativ xotira hajmini ko‘rsatuvchi konsol ilovali dastur tuzilsin.
8. Arifmetik amallarni hisoblash dasturi tuzilsin.
9. Oynada sichqoncha harakatini belgilovchi grafik ilovali dastur tuzilsin.
10. Kvadrat tenglama ildizlarini hisoblovchi dastur tuzilsin.
11. O‘nlik sanoq sistemasida 2 ta sonni o‘zaro qo‘shish dasturi tuzilsin.
12. *ab-cd* ifoda qiymatini hisoblash dasturi tuzilsin.
13. Berilgan sonning kvadrat ildizini hisoblash dasturi tuzilsin.
14. Oynada bir sonning ikkinchi songa nisbatini tasvirlovchi dastur tuzilsin.
15. Windowsda ixtiyoriy matn faylini ochuvchi (ishga tushiruvchi) dastur tuzilsin.

4 – laboratoriya ishi

Mavzu. Assembler tilining makrovositalari bilan ishlash.

Ishning maqsadi. Assembler tilida makrovositalardan foydalanib dasturlar yaratish, makrovosita va protsedura (qism dastur) larning bir-biridan farqli tomonlarini o‘rganish va tahlil qilish bo‘yicha amaliy bilim va ko‘nikmalar hosil qilish.

Masalaning qo‘yilishi. 21h uzilishning 09h va 4ch funksiyalaridan foydalanib, ekranda “Amaliy matematika va informatika” matnini 3 ta satr bilan chiqaruvchi dastur makrovositalar ishtirokida tuzilsin.

Qisqacha nazariy ma’lumot. Assembler tizimning **makrovositalari** deganda dastur matnini shakllantirishning samarali ko‘rinishini taqdim etadigan dastur elementlari to‘plami tushuniladi.

Makrovositalar dasturni assembler tilida yozishni yengillashtirish va bu dastur matnini tushunishni qulaylashtirishga mo‘ljallangan.

Makrovositalardan foydalanishning asosiy mohiyati quyidagicha: agar dastur matni tarkibida bir necha marta takrorlanuvchi fragmentlar (dastur qismlari, bo‘laklari) mavjud bo‘lsa, bu fragment bir marta makros ko‘rinishida yoziladi va unga nom beriladi. Keyin dasturning qayerida unga ehtiyoj sezilsa, xuddi qism dasturdagi (protseduradagi) singari, maxsus buyruqlar yordamida o‘sha joyda unga (makros nomiga) murojaat (ссылка) qilinadi.

Yuqoridagi jarayonni amalga oshirish uchun makrovosita elementlari sifatida xususiy holda makroaniqlov, makrobuyruq va makrokengaytmalardan foydalaniladi.

Ma’lumki, assembler translyatori ikki qismdan iborat: Makrogenerator (makroassembler) va bevosita translyator. Makroassemblerlar bilan ishlashda biz aynan MASM (macroassembler language) sistemasidan foydalanamiz.

Makrovositalardan foydalanilgan dasturlarni translyatsiya qilish jarayoni ikki bosqichda amalga oshiriladi. Birinchi bosqichda makrogenerator ishlaydi, ikkinchi bosqichda makrogenerator yordamida dastlabki matn almashtirishlardan so‘ng hosil bo‘lgan yangi dastur matni translyatsiya qilinadi va obyektli fayl (modul) hosil qilinadi.

Makroaniqlov - bu aslida makrosning yozilishidir. Makroaniqlov sintaksisi quyidagicha:

```
<makrobuyruq nomi> MACRO <formal parametrlar>
... (makroaniqlovning asosiy qismi)
Endm
```

Misol.

```
; satrni ekranga chiqarish
Outstr makro str
    Push ax
    Mov ah 09h
    Lea dx, str
    Int 21h
    Pop ax
    Endm
```

Izoh.

1) dastur tarkibida bir makros nomi ikkinchi marta takrorlanishi mumkin emas;

2) dasturda makros istalgan joyda qo‘yilishi mumkin, lekin u albatta unga murojaat qilish buyrug‘idan oldinda turishi kerak;

3) agar zaruriyat tug‘ilsa, makrosni joriy dasturdan alohida mustaqil fayl shaklida yozib, asosiy dasturda unga maxsus buyruqlar yordamida murojaat qilish mumkin. Masalan agar makros joriy papkada (katalogda) turgan bo‘lsa, u holda unga unga murojaat qilish asosiy dasturda

```
include <fayl nomi>
```

buyrug‘i yordamida amalga oshiriladi.

Misol:

```
Masm
Model small
Include Mymacro.inc
```

Ko‘p hollarda standart makros-fayllar MASM sistemasining include (makroslar kutubxonasi) katalogida joylashgan bo‘ladi, masalan kernel32.inc. Agar asosiy dasturda kernel32.inc fayliga murojat qilish zarur bo‘lsa, u holda bu

```
include c:\masm\include\kernel32.inc
```

buyrug‘i orqali bajariladi.

Makrobuyruqlap - bu makroaniqlovlarga (makrosga) murojaat qilishda foydalaniladigan buyruqlardir.

Makrobuyruq sintaksisi quydagicha bo‘ladi:

<Makros nomi> MACRO [<haqiqiy parametrlar>]

Boshqacha aytganda, makrobuyruqlarning bunday ifodalanishini tegishli makrosni shu joyga qo‘yish uchun makrogeneratorga beriladigan ko‘rsatma, deb tushunish mumkin.

Izoh:

1) haqiqiy parametrlar bir-biridan yo vergul, yoki probel bilan ajratib yozilishi shart;

2) *k* - haqiqiy parametr *k* - formal parametr bilan mos kelishi kerak, ya’ni formal va haqiqiy parametrlar soni o‘zaro teng bo‘lishi kerak;

3) agar haqiqiy parametrlar soni formal parametrlar sonidan ko‘p bo‘lsa, u hisobga olinmaydi;

4) agar haqiqiy parametrlar soni formal parametrlar sonidan kam bo‘lsa, u holda yetishmayotgan haqiqiy parametrlar o‘rniga bo‘sh matn qo‘yish (bo‘sh joy qoldirish) uchun ko‘rsatma beriladi.

Makrobuyruqlar yordamida formal parametrlar bilan haqiqiy parametrlarning o‘rin almashish jarayoniga **makroalmashtirish** deyiladi. Makroalmashtirish natijasida hosil qilingan dastur matniga esa **makrokengaytma** deyiladi.

Shunday qilib, makrogeneratorlarning asosiy faoliyati quyidagilardan iborat:

1) makrogenatorlar makroaniqlovni aynan ko‘rsatilgan joyga o‘rnatadi;

2) makros tarkibidagi barcha formal parametrlarni haqiqiy parametrlar bilan almashtiradi;

3) hosil qilingan makrokengaytmani dasturda makrobuyruq o‘rniga qo‘yadi.

Misol:

```
data
    message db        ’Ikkita son kiriting’, 0Dh,0Ah,’$’
    answer  db 0Dh,0Ah, ‘Natijasi:$’
```

```
repeat  db 0dh,0ah, ‘Davom ettirasizmi? (Y/N)$’
. . .
.code
outstr message ...
outstr answer ...
outstr repeat ...
```

Protsedura va makrovositalarning qiyosiy tahlili.

Takrorlanuvchi amallar (fragmentlar) dasturda ham protsedura, ham makroaniqlov ko‘rinishida yozilishi mumkin. Bunda har ikkala holda ham dastur kodining takrorlanuvchi o‘sha sohasi albatta bir martadan yoziladi va unga murojaat ham bitta buyruq orqali amalga oshiriladi. Ammo translyatsiyadan so‘ng protseduralar dastur matni tarkibida o‘zgarishsiz qoladi, makroaniqlov esa necha marta chaqirilsa, o‘shancha martadasturning ko‘rsatilgan joyiga qo‘yiladi va bu holat dastur matnining kengayishiga olib keladi. Bu esa vaqtni tejash bilan birga xotiradan ko‘proq joyni talab qiladi. Umuman olganda, protsedura va makroaniqlov dasturga qo‘llash va ulardan foydalanish xususiyatlaridan kelib chiqib, quydagi xulosalarni qilish mumkin:

1-xulosa. Protsedurani qo‘llash orqali dasturni ixchamlashtirish hisobidan xotirani tejash mumkin. Protseduraga murojaat qilishda:

a) parametrlarning stek yoki registrlarga yuborilishi bajariladi;

b) qaytish adresi eslab qolinadi;

c) o‘tish ta’minlanadi;

d) protsedura ishining nihoyasida qaytish adresi tiklanadi;

e) stek va registrlar tozalanadi.

Shunday qilib, protsedura ishida vaqt sarfining yuzaga kelishi dastur bajarilishi mobaynida ro‘y beradi, makrovositalarda esa vaqt sarfi translyatsiya vaqtida yuzaga keladi.

2-xulosa. Makrovositalarni qo‘llash dastur bajarilishi jarayonida vaqtni ancha tejashga olib keladi. Agar dasturning takrorlanuvchi sohasida ishtirok etuvchi buyruqlar son jihatlan ancha ko‘p bo‘lsa, uni protsedura ko‘rinishida yozish maqsadga muvofiq bo‘ladi.

Shunday qilib, tuziladigan dasturda agar vaqtni tejash birlamchi masalada turgan bo‘lsa, unga makrovositani qo‘llash kerak, agar birlamchi masalada xotirani tejash turgan bo‘lsa, u holda dasturga protsedurani qo‘llash lozim bo‘ladi.

Endi asosiy masalaga o‘tamiz, ya’ni bu masala shartiga ko‘ra, ekranda ustma-ust bir xil matnni uch marta aks ettirish dasturi tuzilishi kerak. Ushbu masalani assembler tilining makrovositalaridan (makroaniqlovdan) foydalanib amalga oshiramiz.

Dastur matni quyidagicha:

;makros.asm

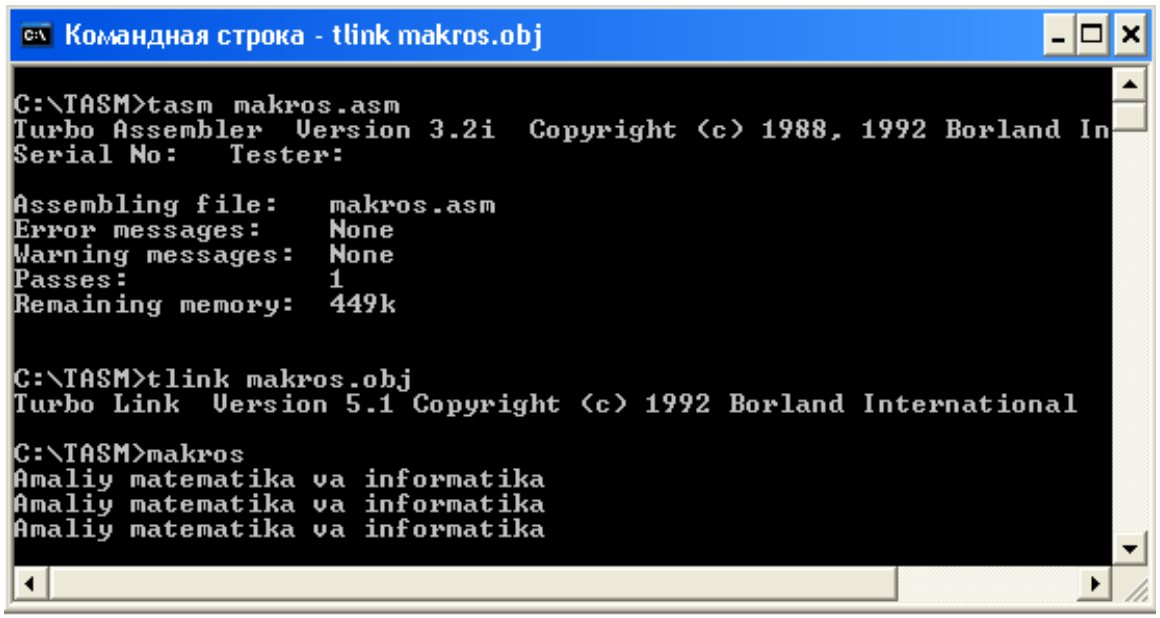
```
.8086
.MODEL  small
.stack 100

.data
hello  db "Amaliy matematika va informatika", 0dh, 0ah, '$'

Quit macro
    mov   ah, 4ch
    int   21h
endm

LDisp macro line
     mov   dx, offset line
     mov   ah, 09
     int   21h
endm

.code
   start:
      mov  dx,@stack
      mov  ss,dx
      mov  dx,@data
      mov  ds,dx
   LDisp hello
   LDisp hello
   LDisp hello
   Quit
   end  start
```

Kompilyatsiyalash algoritmi:

1. TASM uchun:

```
tasm makros.asm
tlink makros.obj
```

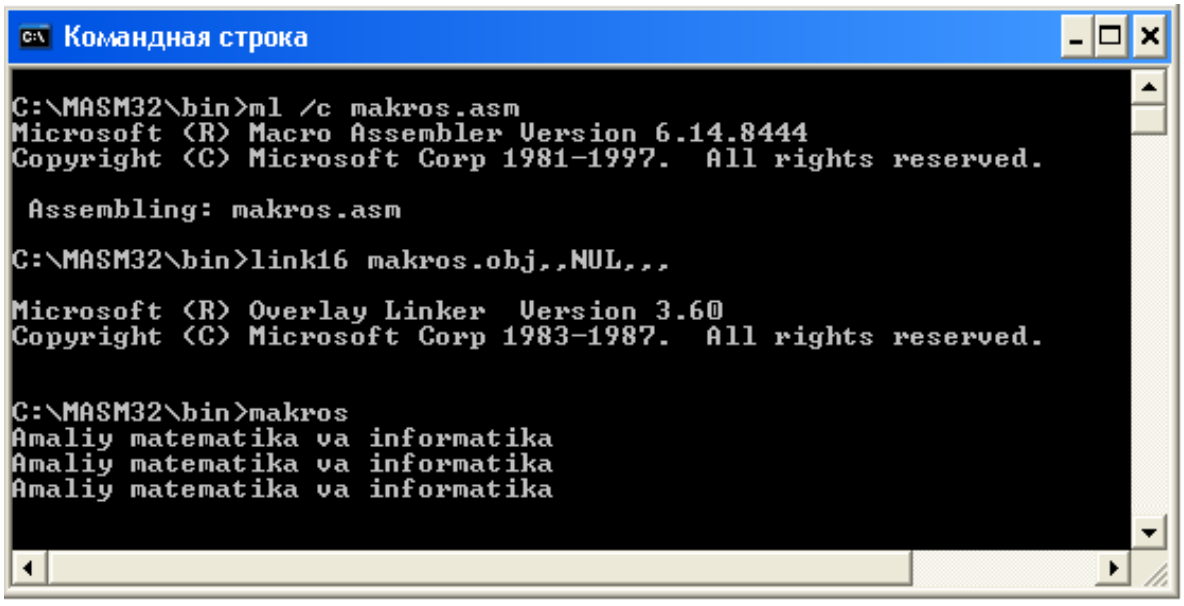
Dastur natijasi:



2. MASM uchun:

```
ml /c makros.asm
link16 makros.obj,,NUL,,,
```

Dastur natijasi:



Xulosa. Tuzilgan dastur 16-bitli rejimga asoslangan bo‘lganligi uchun MASM32 da compilyatsiya qilish jarayonida biz `link16.exe` bog‘lanish muharriridan foydalandik. Tushunishda

qulaylik tug‘dirish maqsadida ushbu sodda ko‘rinishdagi dasturdan foydalanildi. Ammo ko‘pgina matematik ifodalarni hisoblashda bir necha marta takrorlanuvchi ifodalar juda ko‘p uchraydi. Shuningdek, dasturda satrli constantalar o‘rniga belgili constantalarni ham qanday qilib qo‘llash mumkinligini tushunib oldik. Ana shunday masalalarni hal etishda makrosning turli variantlaridan foydalanish mumkin. Makros turlari va undan foydalanish direktivalari ma’ruzalar kursida to‘liqroq yoritilgan.

Topshiriqlar

Assembler tilining makrovositalaridan foydalanib, quyida berilgan topshiriqlarga mos dastur tuzilsin (eslatma tariqasida 1-jadval ilova qilinadi).

1. 21h uzilishning 02h va 08h funksiyalaridan foydalanib, ekranda klavish bosilgandan so‘ng faqat raqam hosil qiluvchi dastur tuzilsin.
2. 21h uzilishning 02h va 08h funksiyalaridan foydalanib, ekranda faqat harfni hosil qiluvchi dastur tuzilsin.
3. 21h uzilishning 02h va 08h funksiyalaridan foydalanib, ekranda lotin alifbosining kichik harflaridan birini hosil qiluvchi dastur tuzilsin.
4. 21h uzilishning 01h va 02h funksiyalaridan foydalanib, ekranda lotin alifbosidagi bosh harflaridan birini hosil qiluvchi dastur tuzilsin.
5. 21h uzilishning 02h va 08h funksiyalaridan foydalanib, ekranda 0-9 raqamlarni ketma-ket satr bo‘yicha hosil qiluvchi dastur tuzilsin.
6. 21h uzilishning 02h va 08h funksiyalaridan foydalanib, ekranda 0-9 raqamlarni ustunma-ustun hosil qiluvchi dastur tuzilsin.
7. 21h. uzilishning 02h va 06h funksiyalaridan foydalanib, ekranda ketma-ket “*” va “!”simvollarni hosil qiluvchi dastur tuzilsin.
8. 21h. uzilishning 02h va 06h funksiyalaridan foydalanib, ekranda “**” va “**!**”simvollarni ustunma-ustun hosil qiluvchi dastur tuzilsin.
9. 21h. uzilishning 0Ah va 09h funksiyalardan foydalanib, ekranda ixtiyoriy satr (matn) hosil qiluvchi dastur tuzilsin.
10. 21h uzilishning 0Ah va 40h funksiyalardan foydalanib, ma’lumotlar segmentiga belgilar satrini joylashtirish va uni ekranda tasvirlash dasturi tuzilsin.

11. 21h uzilishning 0Ah funksiyalardan foydalanib, ma’lumotlar satrida simvolli satrni joylashtiring. So‘ng uni ekranda tasvirlash uchun 21h uzilishning 09h va 40h funksiyalaridan foydalaning.

12. 21h uzilishning 02h va 09h funksiyalardan foydalanib, ma’lumotlar segmentida joylashgan matnni ekranga chiqarish dasturi tuzilsin.

13. 21h uzilishning 02h va 40h funksiyalardan foydalanib, ma’lumotlar segmentida joylashgan belgilarni ekranga chiqarish dasturi tuzilsin.

14. 21h uzilishning 3Fh va 09h funksiyalardan foydalanib, belgilar satrini klaviaturadan kiritish va bu satrni ekranda shu zahoti tasvirlovchi dastur tuzilsin.

15. 21h uzilishning 3Fh va 09h funksiyalardan foydalanib, klaviaturadan ixtiyoriy bitta belgini kiritish orqali uni ekranda hosil qiluvchi dastur tuzilsin.

Int 21h uzilish orqali chaqiriladigan
MS-DOS ning ba’zi asosiy funksiyalari

ilova

H	Kirish	Chiqish	Funksiyasi
01	AL	-	Klavituradan kiritish
02	-	DL	Ekranga 1 ta simvolni chiqarish
05	-	DL	1 ta simvolni printerga chiqarish
06	AL	DL	Boshqaruv pulti orqali bevosita kiritish-chiqarish
07	AL	-	Boshqaruv pul’tidan bevosita kiritish.
08	AL	-	Klaviaturadan kiritish.
09	AL	DX	Ekranga bir necha simvollar ketma-ketligini (satrni) chiqarish
OB	AL	-	Klaviatura holatini tekshirish (o‘qish)
OE	AL	DL	Diskni tanlash.
OF	AL	DX	Fayl ochish.
10	AL	DX	Faylni yopish
14	AL	DX	Ketma-ket o‘qish.
15	AL	DX	Ketma-ket yozish.
1A	-	DX	Buferda 1-buyruq adresini shakllantirish
40	-	DX	Ma’lumotlarni faylga yoki qurilmaga chiqarish

5 - laboratoriya ishi

Mavzu. Dasturni boshqarish. Bir dasturni boshqa dastur orqali ishga tushirish.

Ishning maqsadi. Assembler tilining maxsus boshqaruvchi buyruqlari yordamida Windows amaliy dasturlarini ishga tushiruvchi dasturlar yaratish bo‘yicha amaliy ko‘nikmalar hosil qilish.

Masalaning qo‘yilishi. Assemblerda Windows ning ShellExecute va ExitProcess tizimli funksiyalarini chaqirish orqali MS Word matn muharririni ishga tushiruvchi dastur tuzilsin.

Qisqacha nazariy ma’lumot. Amaliy ish aynan Windows ilovalarini ishga tushiruvchi dasturni yaratishdan iborat bo‘lganligi uchun dasturni bevosita Windows asosida tuzishga harakat qilamiz. Windows oilasiga mansub istalgan operatsion tizim DOS ga qaraganda ancha murakkab bo‘lsada, ular uchun assemblerda dastur tuzish unchalik qiyinchilik tug‘dirmaydi. Birinchidan, Windows ilovalari 32-bitli rejim asosida ishga tushiriladi (flat xotira modeli asosida), ikkinchidan, kompyuter har bir qurilmalari va operatsion tizim har bir moduli uchun quyi darajadagi dasturlash ishlari talab qilinmaydi. Chunki zamonaviy operatsion tizimlar tuzilish tarkibiga ko‘ra o‘z ichiga 2500 ga yaqin tizimli funksiyalardan tashkil topgan dinamik kutubxonalar modulini mujassamlashtirgan (masalan, Windows 95 da 2200 ga yaqin, Windows NT da 2434 ta). Ma’lumki, Win32 da dasturlashda bizga Windows ning tizimli kutubxonani eksport qiluvchi WinAPI vositasi yordamga keladi. API (Application Program Interface) funksiyalarini chaqirish orqali dasturda funksiyaning kirish nuqtasiga boshqarishni uzatishimiz mumkin. Bizga kerakli funksiyalar esa kernell32.dll, user32.dll, gdi32.dll, advapi32.dll yoki boshqa biror kutubxonalarning bittasida joylashgan bo‘lishi mumkin. Qaysi kutubxona fayli bizga kerak bo‘lsa, o‘shani xotiraga yuklashimiz yoki chaqiruv funksiyalari yordamida uni dastur tarkibiga kiritishimiz lozim bo‘ladi. Masalan, bizga hozircha kernell32.dll kerak bo‘lishi muqarrar, chunki kernell32.dll tarkibida xotirani boshqarish, dasturlarni yuklash va o‘chirish bilan bog‘liq bir qator tizimli funksiyalar mavjud.

API funksiyalarini chaqirishning dasturdagi umumiy ko‘rinishi quyidagicha bo‘ladi:

```
Push параметр 3
Push параметр 2
Push параметр 1
Call  функция
```

Win32 uchun tuzilajak har bir dastur oxirida odatga ko‘ra **ExitProcess** tizimli funksiya chaqiriladi, xususan,

```
Call  ExitProcess
```

API – funksiyalarning ikki xil tipi mavjud: ANSI va Unicode. ANSI funksiyalar nomining oxiriga “A” belgisi qo‘yiladi, masalan, ShellExecuteA. Unicode funksiyalar nomining oxiriga esa “U” belgisi qo‘yiladi (chunki, Windows 95 da asosan ANSI funksiyalardan, Windows NT da esa Unicode funksiyalardan foydalaniladi). Asosiy dastur matni tuzilgandan keyin qo‘shimcha ravishda ilova qilinadigan **kernel32.inc** va **shell32.inc** fayllarini yaratishga to‘g‘ri keladi, yuqoridagi tushunchalar bilan biz keyinroq tanishamiz.

Endi asosiy dastur matnini tuzishga o‘tamiz. Dastur tarkibidagi ko‘pgina elementlar haqidagi tushunchalar sizga ma’ruza kurslarida berilgan. Shu sababli bu tushunchalarga to‘xtalmaymiz.

Dastur matni quyidagicha:

;winword.asm

```
include shell32.inc
include kernel32.inc

        .386
        .model     flat
        .const
Path    db 'C:\Program Files\Microsoft Office\Office10\Winword.exe', 0
        .code
start:
        xor        ebx,ebx
        push       ebx
        push       ebx
        push       ebx
        push       offset Path
        push       ebx
        push       ebx
```

```
        call      ShellExecute
        push       ebx
        call      ExitProcess
        end        start
```

Dasturni MASM32 da kompilyatsiya qilishni mo‘ljallaganimiz uchun uni **winword.asm** deb nomlaymiz va **C:\masm32\bin** katalogida saqlaymiz.

Endi bu dasturimizni kompilyatsiya qilishdan oldin bizga hali kerak bo‘ladigan **kernel32.inc** va **shell32.inc** fayllarini yaratamiz**:**

kernel32.inc fayli

```
; kernel32.inc

ifdef _TASM_
        includelib import32.lib
        extrn      ExitProcess:near
else
        includelib kernel32.lib
        extrn      __imp__ExitProcess@4:dword
        ExitProcess  equ  __imp__ExitProcess@4
Endif
```

Bu faylni **kernel32.inc** nomi bilan C:\masm32\bin\ katalogida saqlab qo‘yamiz va keyingi **shell32.inc** faylini yozishga o‘tamiz hamda uni ham aynan shu katalogda saqlaymiz:

shell32.inc fayli

```
; shell32.inc
ifdef   _TASM_
        includelib import32.lib
        extrn   ShellExecuteA:near
        ShellExecute  equ  ShellExecuteA
else
        includelib shell32.lib
        extrn   __imp__ShellExecuteA@24:dword
        ShellExecute  equ  __imp__ShellExecuteA@24
Endif
```

Shunday qilib, bizda **WinWord.asm, kernel32.inc,** va **shell32.inc** dan iborat uchta fayl bor. Endi uni kompilyatsiya qilsak bo‘ladi.

Kompilyatsiyalash algoritmi:

1. Assemblerlash. MS-DOS ni ishga tushirib, avval MASM32, keyin Bin katalogiga o‘tib, undagi **ml.exe** buyrug‘i orqali dasturni assemblerlaymiz. Buning uchun buyruqlar satrida quyidagi buyruqni amalga oshiramiz:

```
ml.exe /c /coff winword.asm
```

Agar dasturda xatolik bo‘lmasa, u holda ekranga quyidagi yozuv chiqadi:

```
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

 Assembling: winword.asm
```

Demak, dastlabki dastur matni asosida obyektli modul (winword.obj) hosil qilindi. Endi ikkinchi qadamni amalga oshirish mumkin, ya’ni **Link.exe** buyrug‘i yordamida bog‘lanish muharririni ishga tushiramiz (uni odatda qisqa qilib linklash deyishadi).

2. Linklash. Buning uchun MS-DOS buyruqlar satriga quyidagilarni kiritamiz:

```
link  /subsystem:windows  winword.obj
```

Buyruq muvaffaqiyatli bajarilganligi va .exe tipli fayl hosil qilinganligini quyidagi xabardan bilish mumkin:

```
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
```

Nihoyat, winword.exe fayli hosil qilindi. Endi uni ishga tushirib ko‘rish mumkin.

Xulosa. Dastur tuzish mobaynida assembler CALL buyrug‘i orqali ikkita tizimli funksiya chaqiruvidan foydalanildi: ShellExicute (faylni ochish) va ExitProcess (jarayonni yakunlash). Shuningdek, 2-baytli Push 0 buyrug‘i o‘rniga 1-baytli Push ebx buyrug‘i ishlatildi. Hosil qilingan faylining hajmi 1536 baytdan iborat!

Topshiriqlar

Mustaqil ravishda yuqoridagi dasturga binoan quyidagi dastur-ilovalarni ishga tushirib ko‘ring va natijasini tahlil qiling

Variant	Vazifa	Variant	Vazifa
1	MS Paint	9	NotePad
2	MS Excel	10	QEditor
3	MS Access	11	Ekran klaviaturasi
4	MS Power Point	12	Ekran lupasi
5	Internet Explorer	13	Tizim kalkulyatori
6	Total Commander	14	Windows Media Player

6 - laboratoriya ishi

Mavzu. COM- va EXE-fayllar bilan ishlash. EXE-faylni COM tipiga o‘tkazish.

Ishning maqsadi. Assembler tilida COM- va EXE-fayllar yaratish, ularni bir-biridan farqli tomonlarini o‘rganish va tahlil qilish, EXE-faylni COM-faylga aylantirish uchun kompilyatsiyalash buyruqlaridan foydalanish qoidalarini o‘rganish bo‘yicha amaliy bilim va ko‘nikmalar hosil qilish.

Masalaning qo‘yilishi. Assemblerda “Bu mening dasturim” matnini ekranga chiqaruvchi COM va EXE tipli dasturlar yaratish misolida dastur tuzilishi va natijasini tahlil qilish.

Qisqacha nazariy ma’lumot. Dastur istalgan operatsion tizimda bajariluvchi va foydalaniluvchi bo‘lishi uchun u aynan bironta bajariluvchi faylga kompilyatsiyalangan bo‘lishi zarur. DOS da bajariluvchi dasturlar asosan ikki tipda bo‘ladi: COM va EXE. Bu ikki tipdagi dasturlarning bir-biridan farqlanuvchi quyidagi jihatlarini ko‘rib chiqamiz:

1. EXE tipli fayllar istalgan o‘lchamga ega bo‘lishi mumkin, ammo COM tipli fayllar hajmi 64 Kb dan oshmaydigan bitta segment o‘lchami bilan chegaralanadi, xolos. Demak, bunday fayllar o‘lchami 64 Kbaytdan yuqori bo‘lmaydi.

2. EXE-faylni yaratishda dastur tarkibida stek segmentini e’lon qilish masalasiga alohida e’tibor qaratiladi, COM-dasturlarda stek segmenti avtomatik ravishda generatsiya qilinadi.

3. EXE-dasturlarda ma’lumotlar segmenti oddiy aniqlanadi va DS registri bu segment adresi bilan faollashtiriladi. COM-dasturda esa bularning barchasi bita kod segmentida aniqlanadi.

4. EXE va COM formatli dasturlarni kompilyatsiya qilishda avval assemblerlash orqali OBJ-fayl olinadi va so‘ngra yig‘uvchi (kompanovshik) LINK dastur buyrug‘i yordamida EXE-fayl hosil qilinadi. Ushbu algoritmlardan so‘ng, agar dastur EXE-faylga mo‘ljallangan bo‘lsa, u holda dastur tayyor, ya’ni uni bajarib ko‘rish mumkin bo‘ladi, aks holda, agar dastur COM-faylga mo‘ljallanib tuzilayotgan bo‘lsa, assemblerlash va linklashdan so‘ng ekranda quyidagi

Warning: No STACK Segment
(Diqqat: Stec segmenti aniqlanmadi)

kabi xabar paydo bo‘ladi. Bunda EXE-fayl qurilgan bo‘lsa-da, ammo bajarilganda kerakli natijani bermaydi. Uni keyingi qadamda COM-faylga o‘tkazish mumkin. Buning uchun MS DOS buyruqlar satrida quyidagi buyruqni terish yetarli:

EXE2BIN Fayl_nomi.exe Fayl_nomi.com

Bu yerda Fayl_nomi yaratilayotgan faylning nomini anglatadi.

5. EXE-fayldan farqli ravishda COM-fayl diskda sarlavhaga ega bo‘lmaydi va u har doim xotiraga yuklanuvchan ko‘rinishda (xususiyatda) bo‘ladi. EXE-fayl esa ikki qismdan tashkil topadi: dasturni boshqarish va sozlash haqidagi axborotlarni o‘z ichiga olgan **sarlavha-yozuvdan** hamda **xususiy yuklanuvchi moduldan**.

Shunday qilib, bu ikki tipdagi fayllar har biri o‘ziga xos rejimda DOS ga xizmat ko‘rsatadi. Umuman olganda, EXE-faylni COM-faylga o‘tkazish mumkin. Buning uchun avvalo dastur tarkibi

tuzilishini loyihalashtirishda amal qilinadigan quyidagi qoidalarga e’tiborni qaratish zarur:

1. Dasturni protsedura ko‘rinishida rasmiylashtirmasdan, uning boshlanishiga START belgisi qo‘yilib, uni END START buyrug‘i bilan tugatish kerak;

2. Dastur boshida xotira modelidan so‘ng ORG 100H buyrug‘ini o‘rnatish kerak. Bu operator kodning berilgan adresdan boshlanishini bildiradi.

3. Dasturni qiymatlar sohasidan boshlash maqsadga muvofiq.

4. MOV AX, @DATA ko‘rinishdagi segmentlarni o‘rnatish mumkin emas. Belgining bittaga siljishini ta’minlash yetarli.

5. Dastlabki kodda stek segmenti to‘liq tushirib qoldiriladi. Stek ko‘rsatuvchisi 64 Kbaytli adres sohasining cho‘qqisiga o‘rnatiladi.

6. Dastur RET ko‘rsatmasi yoki INT 20H uzilish buyrug‘i bilan tugatilishi kerak. 20H uzilish dasturni standart tugatib, boshqarishni operatsion tizimga qaytaradi. Dastur RET ko‘rsatmasi bilan tugatilganda ham aslida 20H uzilish chaqiriladi. Dasturning oxirgi buyrug‘i bajarilgandan keyin RET stek cho‘qqisida turgan 0 (nol) ni siqib chiqaradi va buyruqlar sanagichini PSP ning boshlanishiga o‘rnatadi.

Endi tushunarliroq bo‘lishi uchun ekranga “Bu mening dasturim” matnini hosil qiluvchi COM- hamda EXE-fayllarni bir vaqtda assembler tilida qanday tuzilishini ketma-ket ko‘rib chiqamiz. Dastlab dasturni COM-fayl uchun tuzamiz. Dastur matni quyidagicha:

;d1.asm
;Ekranga “Bu mening dasturim” xabari chiqadi

```
        .model    tiny
        .code
        org       100h
start:
        mov       ah, 9
        mov       dx, offset message
        int       21h
        ret
message db        ‘Bu mening dasturim’, 0Dh,0Ah, ‘$’
        end       start
```

Bu dasturning qanday ishlashini tushunish uchun uning har bir satridagi yozuvlarni birma-bir izohlashga harakat qilamiz.

Dasturning 1-satri **.model tiny** yozuvi bilan boshlangan. Bu xotiraning tiny modelidan foydalanilayotganligini bildiradi. Bu modelga ko‘ra kod, ma’lumotlar va stek segmenti bitta segmentga birlashtirilgan. Bu model COM tipli fayllarni yaratishga mo‘ljallangan.

Dasturda faqat bitta segmentdan foydalanish ko‘zda tutilgan. U ham bo‘lsa kod segmenti. Bu esa dasturimizning ikkinchi satridagi `.code` direktivasi bilan ifodalangan.

Uchinchi satrdagi `org 100h` buyrug‘i dastur hisoblagichi qiymatini 256 bayt (16 lik sanoq tizimida 100H) adresdan boshlab o‘rnatadi. Chunki undan oldingi dastlabki 256 baytni dastur PSP si egallaydi. Shuning uchun COM-faylga kompilyatsiya qilinadigan barcha dasturlar ana shu direktivadan boshlanishi lozim.

Dasturning kodlar bloki start: belgisi bilan boshlanib, u dastur qaysi buyruq bilan boshlanishini bildiradi va oxirida end start buyrug‘i bilan tugatiladi.

Mov ah,9 buyrug‘i AH registrga DOS ning 09H (satrni ekranga chiqarish) funksiyasi nomerini joylashtiradi.

Mov dx, offset message yozuvi bilan ifodalanayotgan buyruq dasturda oxiridan ikkinchi satrdagi message belgisi ostida aniqlanadigan satrni DX registrga joylashtirishni ta’minlaydi.

int 21h buyrug‘i DOS ning tizimli funksiyasini chaqiradi. Bu buyruq yaratilayotgan dastur bilan operatsion tizim o‘rtasidagi asosiy vosita hisoblanadi.

RET buyrug‘i dasturni standart qoidaga ko‘ra tugatib, boshqaruvni yana DOS ga qaytarishni ta’minlaydi.

Navbatdagi satr orqali ekranga chiqarilishi kerak bo‘lgan matn mazmuni message db direktivasi bilan ifodalanadi. Bunda 0Dh (karetkani qaytarish) va 0Ah (satrni o‘girish) kodlar ASCII ning boshqaruvchi simvollari bo‘lib, bu ikki simvol kursorni yangi satrning boshidan o‘rnatilishini ta’minlaydi, **‘$’** belgisi esa satrni tugatuvchi simvol hisoblanadi.

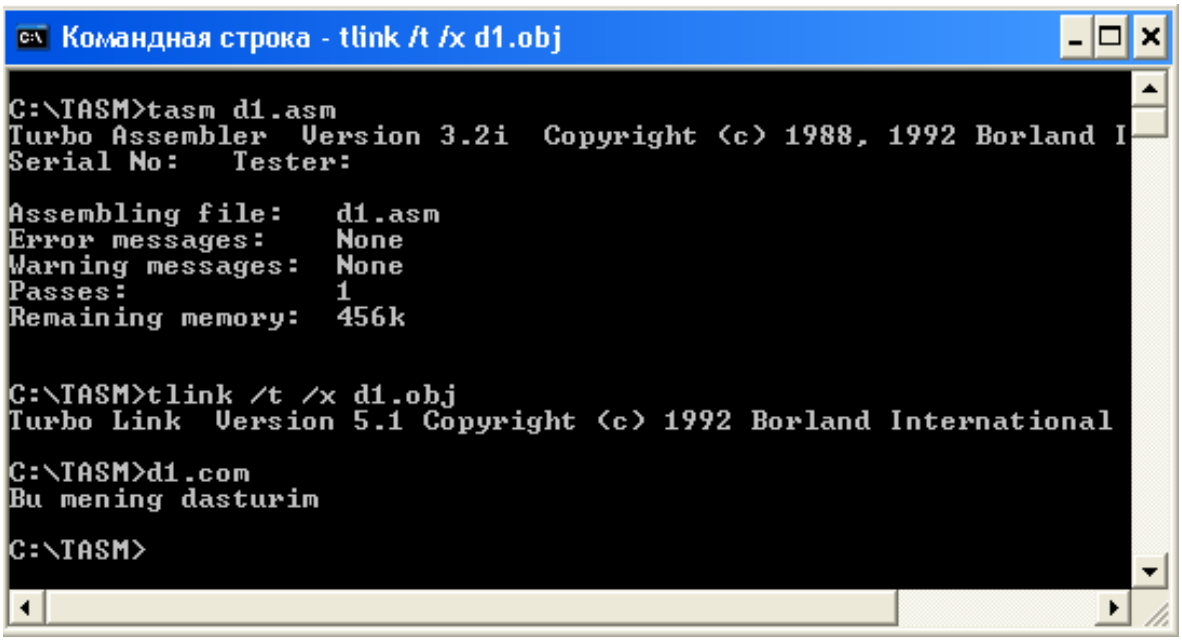
Va, nihoyat dastur tugashini ta’minlash uchun odatga ko‘ra dastu oxirida end start direktivasidan foydalaniladi.

Kompilyatsiyalash algoritmi:

1. TASM uchun:

```
tasm d1.asm
tlink /t /x d1.obj
```

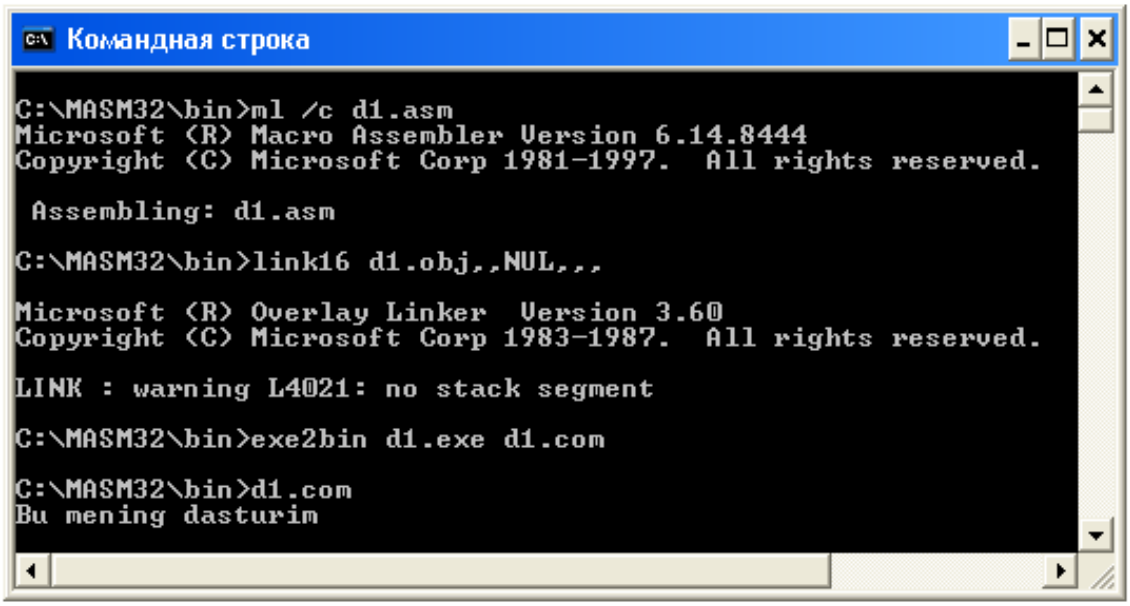
Dastur natijasi:



2. MASM uchun:

```
ml /c d1.asm
link16 d1.obj,,NUL,,,
exe2bin d1.exe d1.com
```

Dastur natijasi:



Endi yuqorida tuzilgan dasturga ekvivalent EXE-dastur matnini keltiramiz (fayl nomini avvalgisidan farqlash uchun d2.asm deb nomlaymiz):

```
;d2.asm
;Ekranga “Bu mening dasturim” xabari chiqadi
        .model    small
        .stack    100h
        .code
start:
        mov       ax,DGROUP
        mov       ds,ax
        mov       dx, offset message
        mov       ah,9
        int       21h
        mov       ax,4C00h
        int       21h
        .data
message db        'Bu mening dasturim',0dh,0ah,'$'
end       start
```

Bu dasturda oldingisidan farqli ravishda xotira modeli sifatida SMALL dan foydalanilgan. Bu modelga ko‘ra kod bir segmentda qolganlari boshqa segmentlarda joylashadi. Assemblerda EXE-dastur tuzishda ushbu modeldan foydalanish ancha qulaylik tug‘diradi.

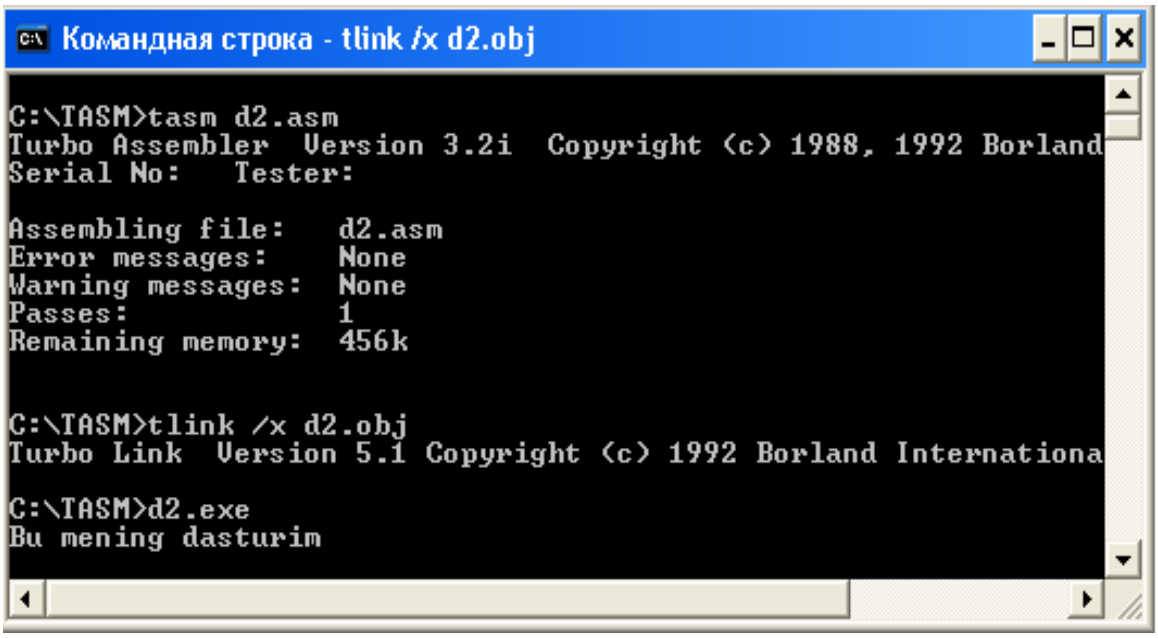
Ko‘rinib turibdiki, bu misolda dastur tarkibida uchta segment aniqlanayapti, ya’ni stek segmenti .stack 100h direktivasi bilan, kod segmenti .code direktivasi bilan va ma’lumotlar segmenti .data direktivasi bilan berilgan. Bunda mov ax,DGROUP buyrug‘i DGROUP ma’lumotlar segmenti guruhining segmentli adresini AX registrga yuklaydi, undan keyingi mov ds,ax buyrug‘i esa uni DS ga nusxalaydi. Dasturni MASM yoki TASM ning keyingi versiyalari muhitida tuzishga to‘g‘ri kelganda, DGROUP o‘rniga @data belgisini qo‘yib ishlatish ham mumkin. Va nihoyat, EXE-dasturlar DOS ning 4C00h tizimli funksiyasini chaqiruv buyrug‘i bilan tugatilishi shart, chunki bunda AH registrda 4ch ning qiymati joylashadi, AL registrda esa 0 ni qaytarish kodi joylashadi (bizning misolimizda bularning barchasi bitta mov ax,4C00h buyrug‘i bilan yuklanadi), undan keyin 21h uzilish chaqiriladi.

Kompilyatsiyalash algoritmi:

1. TASM uchun:

tasm d2.asm
tlink /x d2.obj

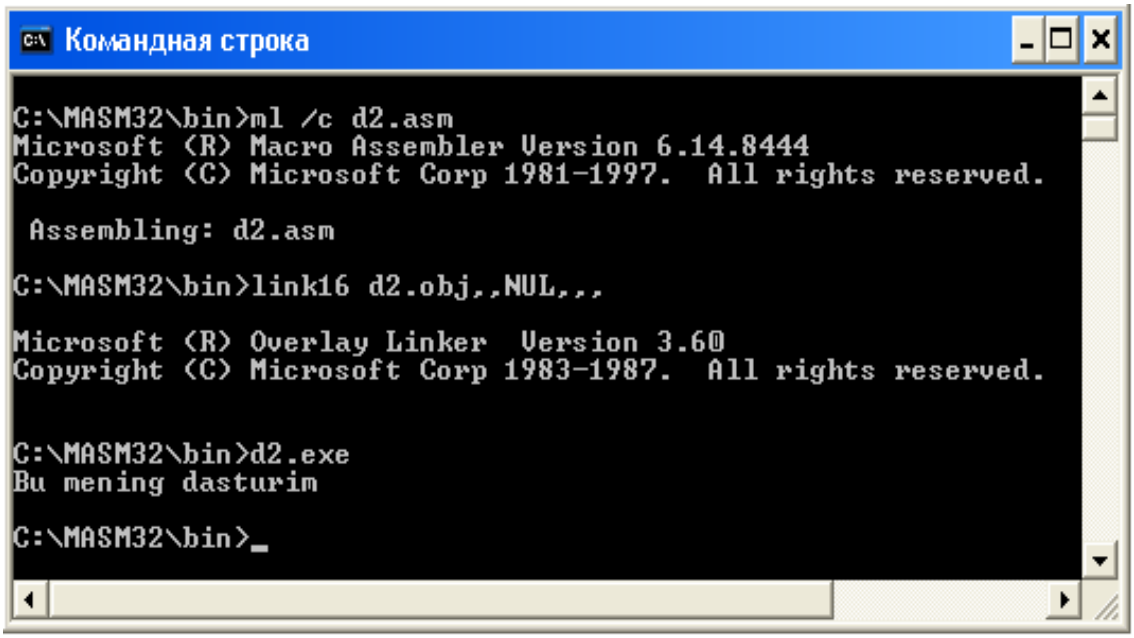
Dastur natijasi:



2. MASM uchun:

ml /c d2.asm
link16 d2.obj,,NUL,,,

Dastur natijasi:



Xulosa. Tuzilgan dasturlarning natijalariga ko‘ra **d1.com** faylning o‘lchami TASM da ham, MASM da ham kompilyatsiya

qilinganiga qaramasdan 29 baytni, **d2.exe** faylning o‘lchami esa TASM da 565 baytni, MASM da esa 551 baytni tashkil qildi.

Topshiriqlar

Quyida berilgan topshiriqlarga mos COM- va EXE-dasturlar bir vaqtda tuzilsin va dastur natijalari tahlili hisobotda talab qilingan bandlar bo‘yicha yozma shaklda keltirilsin.

Вариант	Vazifa
1	RTC dan joriy sanani aniqlash va ekranga chiqarish
2	RTC dan tizimli soatni o‘qish va ekranga chiqarish
3	Ekrandagi mavjud tasvir nusxasini bosmaga chiqarish
4	Operativ xotira hajmini aniqlash
5	Disket (A:) ga belgi (Label) qo‘yish
6	Disket (A:) parametrlarini aniqlash
7	Tizimni qayta yuklash
8	ASCII belgilarini ekranga chiqarish
9	Qattiq disk (C:) parametrlarini aniqlash
10	Dasturni rezident holda qoldirish
11	Ekranga tez chiqarishni ta’minlash
12	Klaviatura Ctrl-Break tugmasi bosilishini qayta ishlash
13	Klaviatura Ins tugmasi bosilishini qayta ishlash
14	Klaviatura Caps Look tugmasi bosilishini qayta ishlash
15	Klaviatura Num Look tugmasi bosilishini qayta ishlash

7- Laboratoriya ishi

Mavzu: Tizim uzilishlari. DOS va BIOS uzilishlari bilan ishlash.

Ishning maqsadi. DOS va BIOS funksiyalaridan foydalanib, tizim chaqiruvlarini hosil qilish, uzilish turlari va mohiyatini chuqurroq o‘rganish yuzasidan amaliy ko‘nikmalar hosil qilish.

Masalaning qo‘yilishi: BIOS 10h uzilishining 06h funksiyasi, BIOS 10h uzilishi 13h funksiyasi va DOS 21h uzilishining 09h funksiyasi bilan ishlashga doir dasturlar tuzish.

Qisqacha nazariy ma’lumot. Kompyuter ishlashi jarayonida yuzaga kelgan nostandart holatlarni bartaraf etish maqsadida tizim chaqirishlari yoki uzilishlar deb ataluvchi dasturiy mexanizimlardan foydalaniladi. Bu mexanizimga ko‘ra, kompyuterning bajarib turgan joriy ishi qanday bo‘lishidan qat’iy nazar, ma’lum muddatga yuborilgan signal bu faoliyatga vaqtincha chek qo‘yadi va yangi dastur o‘z ishini tugatib bo‘lgach, kompyuterga hech narsa bo‘lmagandek, yana aavalgi ishini davom ettiraveradi. Demak, bir tomonlama qaraganda, **uzilish** - bu boshqa protsedura bajarilayotgan vaqtda joriy protseduraning vaqtinchalik to‘xtab turish jarayonidir. Ikkinchi tomondan, uzilish – bu biror dastur bajarilayotgan paytda aynan mo‘ljaldagi masalani hal etish maqsadida chaqiriladigan tayyor dastur-protseduradir.

Uzilishlarni keltirib chiqaruvchi manbaga nisbatan ularni asosiy uchta guruhga ajratish mumkin:

1. Zahiraviy uzilishlar (tashqi yoki apparat uzilishlari deb ham ataladi);
2. Dasturiy uzilishlar (ichki uzilishlar deb ham ataladi)
3. Mantiqiy uzilishlar (protsessor uzilishlari deb ham ataladi).

Zahiraviy uzilishlar asosan kompyuterning turli qurilmalaridan keladigan signallar orqali hosil bo‘ladi. Masalan, elektr tarmog‘ida kuchlanishning tushishi, klaviatura tugmalarining bosilishi, tizimli vaqt o‘lchagichdan navbatdagi impulsning uzatilishi va hokazo.

Dasturiy uzilishlar asosan dastur tomonidan yuzaga keltiriladi, bu holat bir dasturga ikkinchi dastur tomonidan xizmat ko‘rsatilishi talab qilingan vaziyatlarda paydo bo‘lishi mumkin.

Mantiqiy uzilishlar asosan mikroprotsessor ish faoliyatida yuzaga keladigan nostandart holatlar ta’sirida vujudga keladi, masalan, nolga bo‘lish amali, registrlarning to‘lishi hodisasi va boshqalar.

Umuman olganda, har bir guruhdagi uzilishlarning o‘ziga xas tartib raqamlari mavjud. Shaxsiy kompyuterlarda bir-biridan farqli 256 xil uzilish mavjud bo‘lib, ular 0 dan 255 gacha (16 lik sanoq tizimida bu 0 dan ff gacha) bo‘lgan raqamlar orqali belgilanadi. Bu uzilishlarning 8 tasi (00h-07h) markaziy protsessor uzilishlariga, 8 tasi

(08h-0Fh) 1-saviyali uzilish kontrollyorlaridan bo‘ladigan uzilishlarga, 16 tasi (10h-1Fh) BIOS uzilishlariga, 32 tasi (20h-3Fh) DOS uzilishlariga, 17 tasi (40h-50h) qo‘shimcha to‘ldirilgan BIOS uzilishlariga, 15 tasi (51h-5Fh) tarmoq uzilishlariga, 16 tasi (60h-6Fh) foydalanuvchi uzilishlariga, 8 tasi (70h-77h) 2-saviyali uzilish kontrollyorlaridan keladigan uzilishlarga, qolgan 136 tasi (78h-FFh) turli maqsadli boshqa uzilishlarga kiradi.

Misol tariqasida BIOS tomonidan displeyni boshqarishga mo‘ljallangan 10h uzilishining 06h funksiyasidan foydalanilgan ushbu dasturni ko‘rib chiqamiz. Ma’lumki, 10h uzilishining 06h funksiyasi MS DOS ning CLS buyrug‘iga ekvivalent bo‘lgan ekranni tozalash vazifasini bajaradi.

Dastur matni quyidagicha:

;1-m.asm

```
;ekranni tozalovchi dastur

   .model tiny
     .code
     org 100h
start:
     push AX
     push BX
     push CX
     push DX
     mov AH,6
     xor AL,AL
     mov BH,00000111b
     xor CX,CX
     mov DH,24d
     mov DL,79d
     int 10h
     pop DX
     pop CX
     pop BX
     pop AX
     ret
     end    start
```

Dasturni kompilyatsiyalash algoritmi:

```
tasm /m  1-m.asm
tlink /t /x 1-m.obj
```

Natijada hosil bo‘lgan 1-m.com faylini ishga tushirib, o‘zingiz tahlil qilib ko‘rishingiz mumkin.

Shuningdek, BIOS 10h uzilishi 13h funksiyasi va boshqa bir qator funksiyalari bilan yaqinroq tanishish uchun quyidagi dasturga e’tibor bering va kompilyatsiyalash natijasidan so‘ng hosil bo‘lgan bajariluvchi faylni ishga tushirib, o‘zingiz tahlil qilib ko‘ring:

;2-m.asm

```
;ekranga Hello World yozuvi turli ranglarda chiqadi

assume CS:SUXXX, ES:SUXXX
SUXXX segment
org 100h
MAIN proc
  lea bp,ABC
  mov AH,13h
  mov AL,3
  xor bh,bh
  mov bl,07h
  mov cx,16d
  xor dx,dx
  int 10h
  int 20h
MAIN endp
ABC db 'H',0Ah,'e',0Bh,'l',0Dh,'l',0Ch
    db 'o',0Bh,',',0Ah,' ',0Ah,'W',09h
    db 'o',08h,'r',07h,'l',06h,'d',05h
    db '!',02h,'!',02h,'!',02h
SUXXX ends
end MAIN
```

Dasturni kompilyatsiyalash algoritmi 1-misoldagidek bo‘ladi.

Yuqoridagi dasturlardan ham ko‘rinib turibdigi, har bir uzilish maxsus int n mashina buyrug‘i orqali chaqirilib, bu uzilish funksiyalarini belgilangan registrlarga uzatish uchun

```
mov <регистр> <функция>
```

buyrug‘idan foydalaniladi. Ayni shu int n mashina buyrug‘ining o‘zi ham bir dasturiy uzilishdir. Tushunish osonroq bo‘lish uchun dasturiy uzilishga misol sifatida DOS 21h uzilish va uning ba’zi funksiyalari haqida to‘xtalamiz. DOS 21h uzilishi dasturchiga operatsion tizim tomonidan turli xizmatlarni ko‘rsatishga mo‘ljallangan bo‘lib, bu xizmatlarning tegishli DOS funksiyalari orqali beriladi. Masalan,

Int 21h 00h – dastur ishini tugatish;
Int 21h 01h – simvollarni kiritish;
Int 21h 09h – simvollarni ekranga chiqarish;
Int 21h 0Bh – klaviatura buferi holatini tekshirish va hk.

Endi 21h uzilishning 09h funksiyasi bilan ishlashga doir dastur namunasini keltiramiz:

```
;3-m.asm
.model      small
.stack      100h
.data
message1 db     'Bu 1-satr', 0dh,0ah, '$'
message2 db     'Bu 2-satr', 0dh,0Ah, '$'
.code
start:
        mov  ax,@data
        mov  ds,ax
        mov  ah,09h
        mov  dx,offset message1
        int     21h
        mov  dx,offset message2
        int     21h
        mov  ax,4c00h   ; standart chiqish - ah=00h
        int     21h
end   start
```

Kompilyatsiyalash:

```
tasm   3-m.asm
tlink  3-m.obj
```

Natijada bajarilishga tayyor bo‘lgan **3-m.exe** fayli hosil bo‘ladi.

Topshiriqlar

Ushbu masalalar qaysi turdagi uzilishga tegishli ekanligi aniqlansin va bu uzilishlarga mos dastur tuzilsin:

Variant	Vazifa
1	RTC dan joriy sanani aniqlash va ekranga chiqarish
2	RTC dan tizimli soatni o‘qish va ekranga chiqarish
3	Ekrandagi mavjud tasvir nusxasini bosmaga chiqarish
4	Operativ xotira hajmini aniqlash
5	Disket (A:) ga belgi (Label) qo‘yish
6	Disket (A:) parametrlarini aniqlash
7	Tizimni qayta yuklash
8	ASCII belgilarini ekranga chiqarish
9	Qattiq disk (C:) parametrlarini aniqlash
10	Dasturni rezident holda qoldirish
11	Ekranga tez chiqarishni ta’minlash
12	Klaviatura Ctrl-Break tugmasi bosilishini qayta ishlash
13	Klaviatura Ins tugmasi bosilishini qayta ishlash
14	Klaviatura Caps Look tugmasi bosilishini qayta ishlash
15	Klaviatura Num Look tugmasi bosilishini qayta ishlash